

La guida completa





KEVIN LONEY, GEORGE KOCH

# LA GUIDA COMPLETA ORACLE9I

**McGraw-Hill**

Milano New York San Francisco Washington, D.C. Auckland Bogotá Lisbon London  
Madrid Mexico City Montreal New Delhi San Juan Singapore Sydney Tokyo Toronto

- EDITOR: Maria Grazia Viscito  
• PRODUZIONE: Gino La Rosa  
• REALIZZAZIONE: Studio GSG – Traduttori associati – Lesa (NO)  
• TRADUZIONE: Stefano Gubian, Valeria Cardano  
• STAMPA: Delta Grafica – Città di Castello (PG)
- •  
•  
•  
•  
•

Titolo originale: *Oracle9i: The Complete Reference*

Copyright: © 2002 by The McGraw-Hill Companies

Copyright: © 2003 The McGraw-Hill Companies, S.r.l.

Publishing Group Italia

Via Ripamonti 89 – 20139 Milano

**McGraw-Hill**  
*A Division of The McGraw-Hill Companies* 

I diritti di traduzione, di riproduzione, di memorizzazione elettronica e di adattamento totale e parziale con qualsiasi mezzo (compresi i microfilm e le copie fotostatiche) sono riservati per tutti i Paesi.

Ogni cura è stata posta nella creazione, realizzazione, verifica e documentazione dei programmi contenuti in questo libro, tuttavia né l'Autore né The McGraw-Hill Companies, S.r.l. Publishing Group Italia possono assumersi alcuna responsabilità derivante dall'implementazione dei programmi stessi, né possono fornire alcuna garanzia sulle prestazioni o sui risultati ottenibili dall'utilizzo dei programmi. Inoltre, date le caratteristiche intrinseche di Internet, non sono responsabili per eventuali variazioni negli indirizzi o nei contenuti dei siti riportati. Lo stesso dicasi per ogni persona o società coinvolta nella creazione, nella produzione e nella distribuzione di questo libro.

Nomi e marchi citati nel testo sono generalmente depositati o registrati dalle rispettive case produttrici.

Printed in Italy  
1234567890DELDEL076543  
ISBN 88 386 4307-5

## Indice

<b>Prefazione</b>	XV
<b>Introduzione</b>	XIX
<hr/>	
<b>PARTE PRIMA ● CONCETTI FONDAMENTALI DEI DATABASE</b>	1
<hr/>	
Capitolo 1 <b>Condividere conoscenze e successo</b>	3
1.1    L'approccio cooperativo	4
1.2    Tutti hanno dei "dati"	5
1.3    Il linguaggio familiare di Oracle	6
1.4    Alcuni esempi comuni	12
<hr/>	
Capitolo 2 <b>I pericoli di un database relazionale</b>	15
2.1    È davvero facile come sembra?	15
2.2    Quali sono i rischi?	16
2.3    L'importanza della nuova visione	17
2.4    Codici, abbreviazioni e standard di denominazione	18
2.5    Come arginare la confusione	19
2.6    Maiuscole e minuscole nei nomi e nei dati	26
2.7    Normalizzazione dei nomi	27
2.8    L'importanza del fattore umano	27
2.9    Comprensione dei dati	32
2.10    Verso la normalizzazione dei nomi degli oggetti	36
2.11    Chiavi intelligenti e valori di colonna	38
2.12    Il decalogo	39

Capitolo 3	<b>Le parti fondamentali del discorso in SQL</b>	<b>41</b>
3.1	Stile	42
3.2	Creazione della tabella GIORNALE	43
3.3	Uso di SQL per selezionare dati da tabelle	44
3.4	I comandi select, from, where e order by	47
3.5	Logica e valore	49
3.6	Un altro impiego delle subquery con where	57
3.7	Combinazione di tabelle	61
3.8	Creazione di una vista	63
Capitolo 4	<b>Elementi di base dei database relazionali a oggetti</b>	<b>67</b>
4.1	È obbligatorio utilizzare gli oggetti?	67
4.2	Perché utilizzare gli oggetti?	68
4.3	Tutti possiedono degli oggetti	69
4.4	Un esempio di oggetto comune	72
4.5	Analisi e progettazione orientata agli oggetti	78
4.6	I prossimi capitoli	79
Capitolo 5	<b>Introduzione ai database con capacità Web</b>	<b>81</b>
5.1	Dove si inserisce SQL?	82
5.2	Dove si inserisce Java?	83
5.3	Dove si inserisce Oracle Portal?	84
<b>PARTE SECONDA ● SQL E SQL*PLUS</b>		<b>85</b>
Capitolo 6	<b>Report e comandi fondamentali di SQL*PLUS</b>	<b>87</b>
6.1	Creazione di un report semplice	89
6.2	Altre caratteristiche	99
6.3	Controllo dell'ambiente SQLPLUS	105
6.4	I fondamenti	107
Capitolo 7	<b>Ottenere e modificare informazioni di testo</b>	<b>109</b>
7.1	Tipi di dati	109
7.2	Che cos'è una stringa?	110
7.3	Notazione	110
7.4	Concatenazione (  )	112
7.5	Come tagliare e incollare le stringhe	113
7.6	Le funzioni di stringa con order by e where	129
7.7	Conclusioni	133
Capitolo 8	<b>Giocare con i numeri</b>	<b>135</b>
8.1	Le tre classi di funzioni numeriche	135
8.2	Notazione	136

---

8.3	Funzioni per valori singoli	139
8.4	Funzioni di gruppo	146
8.5	Funzioni di elenco	153
8.6	Ricerca di righe con MAX o MIN	154
8.7	Precedenza e parentesi	156
8.8	Conclusioni	157
 Capitolo 9 <b>Le date</b>		 <b>159</b>
9.1	Aritmetica delle date	159
9.2	ROUND e TRUNC nei calcoli delle date	168
9.3	Formattazione di TO_DATE e TO_CHAR	169
9.4	Date nelle clausole where	178
9.5	Gestione di più secoli	179
9.6	Uso della funzione EXTRACT	180
9.7	Uso dei tipi di dati TIMESTAMP	181
 Capitolo 10 <b>Funzioni di conversione e trasformazione</b>		 <b>183</b>
10.1	Funzioni di conversione elementari	185
10.2	Funzioni di conversione specializzate	190
10.3	Funzioni di trasformazione	190
10.4	Conclusioni	193
 Capitolo 11 <b>Raggruppamento di righe</b>		 <b>195</b>
11.1	Uso di group by e having	195
11.2	Viste di gruppi	199
11.3	La potenza delle viste di gruppi	201
11.4	Conclusioni	206
 Capitolo 12 <b>Quando una query dipende da un'altra</b>		 <b>207</b>
12.1	Subquery avanzate	207
12.2	Outer join	212
12.3	Join naturali e interni	218
12.4	UNION, INTERSECT e MINUS	219
 Capitolo 13 <b>Alcune possibilità complesse</b>		 <b>223</b>
13.1	Raggruppamenti complessi	223
13.2	Uso di tabelle temporanee	225
13.3	Uso di ROLLUP, GROUPING e CUBE	225
13.4	Alberi genealogici e connect by	229
 Capitolo 14 <b>Creazione di un report in SQL*PLUS</b>		 <b>239</b>
14.1	Formattazione avanzata	239
14.2	set termout off e set termout on	252

	14.3	Variabili in SQLPLUS	252
	14.4	Formattazione di numeri	255
	14.5	Uso di mask.sql	256
	14.6	show all e spool	257
	14.7	Aggiunta di linee vuote	257
	14.8	Ulteriori controlli per la realizzazione di report	259
Capitolo 15		<b>Modifica dei dati: insert, update, merge e delete</b>	<b>261</b>
	15.1	insert	261
	15.2	rollback, commit e autocommit	264
	15.3	Inserimenti in più tabelle	267
	15.4	delete	271
	15.5	update	272
	15.6	Uso del comando merge	274
Capitolo 16		<b>Uso avanzato di funzioni e variabili</b>	<b>277</b>
	16.1	Funzioni in order by	277
	16.2	Diagrammi a barre e grafici	278
	16.3	Uso di TRANSLATE	280
	16.4	Copia e incollamento complessi	281
	16.5	Conteggio delle occorrenze di stringhe in stringhe più lunghe	285
	16.6	Ulteriori informazioni sulle variabili	287
Capitolo 17		<b>DECODE e CASE: if, then ed else in SQL</b>	<b>289</b>
	17.1	if, then, else	289
	17.2	Sostituzione di valori con DECODE	292
	17.3	DECODE in DECODE	293
	17.4	Greater Than e Less Than in DECODE	296
	17.5	Uso di CASE	298
Capitolo 18		<b>Creazione, eliminazione e modifica di tabelle e viste</b>	<b>301</b>
	18.1	Creazione di una tabella	301
	18.2	Eliminazione di tabelle	309
	18.3	Modifica di tabelle	309
	18.4	Creazione di una vista	314
	18.5	Creazione di una tabella a partire da un'altra	316
	18.6	Creazione di una tabella di solo indice	318
	18.7	Uso di tabelle partizionate	319
	18.8	Ridefinizione in linea delle tabelle	324

---

Capitolo 19	<b>Oracle e l'autorità</b>	<b>329</b>
19.1	Utenti, ruoli e privilegi	329
19.2	Che cosa possono concedere gli utenti	336
19.3	Concessione di risorse limitate	349
19.4	Opzioni avanzate	349
Capitolo 20	<b>Modifica degli ambienti di Oracle</b>	<b>351</b>
20.1	Indici	351
20.2	Tablespace e struttura del database	358
20.3	Segmenti di rollback e undo gestiti a livello di sistema	360
20.4	Cluster	361
20.5	Sequenze	362
Capitolo 21	<b>Uso di SQL*Loader per caricare i dati</b>	<b>365</b>
21.1	Il control file	366
21.2	Avvio del caricamento	367
21.3	Note sulla sintassi del control file	371
21.4	Gestione delle operazioni di caricamento dei dati	373
21.5	Messa a punto delle operazioni di caricamento dei dati	375
21.6	Altri miglioramenti di Oracle9i	377
Capitolo 22	<b>Accesso a dati remoti</b>	<b>379</b>
22.1	Database link	379
22.2	Uso di sinonimi per la trasparenza di dislocazione	386
22.3	Uso della pseudocolonna User nelle viste	387
22.4	Collegamenti dinamici: uso del comando copy di SQLPLUS	389
22.5	Connessione a un database remoto	390
Capitolo 23	<b>Uso delle viste materializzate</b>	<b>393</b>
23.1	Funzionalità	393
23.2	Privilegi di sistema richiesti	394
23.3	Privilegi di tabella richiesti	394
23.4	Viste materializzate di sola lettura e viste materializzate aggiornabili a confronto	395
23.5	Sintassi di create materialized view	396
23.6	Aggiornamento delle viste materializzate	402
23.7	Sintassi di create materialized view log	410
23.8	Modifica dei log e delle viste materializzate	413
23.9	Eliminazione di viste materializzate e log	414

Capitolo 24	<b>Uso di Oracle Text per le ricerche di testo</b>	<b>415</b>
24.1	Aggiunta di testo al database	415
24.2	Query di testo e indici di testo	416
24.3	Gruppi di indici	429
Capitolo 25	<b>Uso di tabelle esterne</b>	<b>431</b>
25.1	Accesso ai dati esterni	431
25.2	Creazione di una tabella esterna	432
25.3	Limitazioni, vantaggi e usi possibili delle tabelle esterne	440
Capitolo 26	<b>Uso delle query flashback</b>	<b>443</b>
26.1	Esempi di flashback basati sull'ora	444
26.2	Come salvare i dati	445
26.3	Esempi di flashback basati su SCN	447
<b>PARTE TERZA ● PL/SQL</b>		<b>449</b>
Capitolo 27	<b>Introduzione a PL/SQL</b>	<b>451</b>
27.1	PL/SQL: nozioni di base	451
27.2	La sezione delle dichiarazioni	452
27.3	La sezione dei comandi eseguibili	455
27.4	La sezione di gestione delle eccezioni	466
Capitolo 28	<b>I trigger</b>	<b>469</b>
28.1	Privilegi di sistema richiesti	469
28.2	Privilegi di tabella richiesti	470
28.3	Tipi di trigger	470
28.4	Sintassi dei trigger	472
28.5	Attivazione e disattivazione dei trigger	484
28.6	Sostituzione di trigger	485
28.7	Eliminazione di trigger	485
Capitolo 29	<b>Procedure, funzioni e package</b>	<b>487</b>
29.1	Privilegi di sistema richiesti	488
29.2	Privilegi di tabella richiesti	490
29.3	Procedure e funzioni	490
29.4	Procedure e package	490
29.5	Sintassi del comando create procedure	490
29.6	Sintassi del comando create function	492
29.7	Sintassi del comando create package	499

---

29.8	Visualizzazione del codice sorgente di oggetti procedurali	502
29.9	Compilazione di procedure, funzioni e package	503
29.10	Sostituzione di procedure, funzioni e package	504
29.11	Eliminazione di procedure, funzioni e package	504

---

**PARTE QUARTA ● DATABASE RELAZIONALI A OGGETTI** **507**

---

Capitolo 30	<b>Implementazione di tipi, viste oggetto e metodi</b>	<b>509</b>
30.1	Nuova analisi dei tipi di dati astratti	509
30.2	Implementazione di viste oggetto	514
30.3	Metodi	520
Capitolo 31	<b>Collettori (tabelle annidate e array variabili)</b>	<b>525</b>
31.1	Array variabili	525
31.2	Tabelle annidate	531
31.3	Problemi nella gestione di tabelle annidate e array variabili	535
Capitolo 32	<b>Uso di dati LOB</b>	<b>539</b>
32.1	Tipi di dati disponibili	539
32.2	Definizione dei parametri di memorizzazione per i dati LOB	541
32.3	Gestione e selezione dei valori LOB	543
Capitolo 33	<b>Concetti avanzati di orientamento agli oggetti</b>	<b>563</b>
33.1	Oggetti riga e oggetti colonna	563
33.2	Tabelle oggetto e identificativi di oggetto (OID)	564
33.3	Viste oggetto con REF	571
33.4	PL/SQL a oggetti	576
33.5	Oggetti nel database	578

---

**PARTE QUINTA ● JAVA IN ORACLE** **579**

---

Capitolo 34	<b>Un'introduzione a Java</b>	<b>581</b>
34.1	Confronto tra Java e PL/SQL: introduzione	581
34.2	Introduzione	582
34.3	Dichiarazioni	582
34.4	Comandi eseguibili	583
34.5	Classi	591

---

Capitolo 35	<b>Programmazione JDBC e SQLJ</b>	<b>597</b>
35.1	Introduzione	597
35.2	Uso delle classi JDBC	599
35.3	SQLJ	605
35.4	Uso delle classi SQLJ	607
Capitolo 36	<b>Stored procedure Java</b>	<b>613</b>
36.1	Caricamento della classe nel database	614
36.2	Come accedere alla classe	615
<b>PARTE SESTA ● GUIDE</b>		<b>621</b>
Capitolo 37	<b>Guida al dizionario dati di Oracle9i</b>	<b>623</b>
37.1	Nota sulla nomenclatura	624
37.2	Le “carte stradali”: DICTIONARY (DICT) e DICT_COLUMNS	624
37.3	Oggetti da cui è possibile selezionare: tabelle (e colonne), viste, sinonimi e sequenze	625
37.4	Vincoli e commenti	634
37.5	Indici e cluster	639
37.6	Tipi di dati astratti, strutture correlate a ORDBMS e LOB	643
37.7	Database link e viste materializzate	646
37.8	Trigger, procedure, funzioni e package	649
37.9	Dimensioni	652
37.10	Allocazione e uso dello spazio, comprese partizioni e partizioni secondarie	653
37.11	Utenti e privilegi	659
37.12	Ruoli	661
37.13	Audit	662
37.14	Varie	663
37.15	Monitoraggio: le tabelle di prestazioni dinamiche V\$	663
Capitolo 38	<b>Guida all'ottimizzatore di Oracle</b>	<b>669</b>
38.1	Quale ottimizzatore?	669
38.2	Operazioni di accesso alle tabelle	671
38.3	Operazioni che utilizzano gli indici	673
38.4	Operazioni che gestiscono insiemi di dati	682
38.5	Operazioni che eseguono join	692
38.6	Visualizzazione del percorso di esecuzione	702
38.7	Operazioni varie	709
38.8	Conclusioni	715

---

Capitolo 39	<b>Guida a Oracle9iAS</b>	<b>717</b>
39.1	Communication Services	719
39.2	Content Management Services	722
39.3	Ultra Search	723
39.4	Business Logic Services	724
39.5	Presentation Services	725
39.6	Business Intelligence Services	727
39.7	Portal Services	727
39.8	Kit per sviluppatori	729
39.9	Caching Services	730
39.10	System Services	733
39.11	Database Services	734
39.12	Avvio, arresto e riavvio di iAS (Apache)	735
Capitolo 40	<b>Guida all'amministrazione del database</b>	<b>739</b>
40.1	Creazione di un database	739
40.2	Avvio e arresto del database	740
40.3	Ridimensionamento e gestione delle aree di memoria per il database	742
40.4	Allocazione e gestione dello spazio per gli oggetti	743
40.5	Creazione e gestione dei segmenti di rollback	754
40.6	Esecuzione dei backup	757
40.7	Riepilogo	772
Capitolo 41	<b>Guida a XML in Oracle</b>	<b>773</b>
41.1	Document Type Definition, elementi e attributi	773
41.2	XML Schema	776
41.3	Uso di XSU per selezionare, inserire, aggiornare e rimuovere valori XML	779
41.4	Uso di XMLType	784
41.5	Altre caratteristiche	786
Capitolo 42	<b>Guida di riferimento alfabetica</b>	<b>789</b>
42.1	Contenuti della guida di riferimento	789
42.2	Che cosa non contiene la guida di riferimento	789
42.3	Formato generale delle voci	790
42.4	Ordine delle parole chiave	792
	<b>Indice analitico</b>	<b>1173</b>



# Prefazione

## L'interessante storia di questo libro

Il mio primo incontro con Oracle risale al 1982, durante la valutazione di diversi sistemi di gestione di database per un'importante applicazione commerciale che la mia società si stava preparando a progettare e sviluppare. Al termine, il nostro lavoro di analisi venne definito dalla rivista *ComputerWorld* il più "estenuante" studio sui DBMS che fosse mai stato condotto in assoluto. L'indagine era così severa nei confronti dei diversi produttori dei programmi esaminati che riuscì a far parlare di sé persino la stampa della Nuova Zelanda e a suscitare l'interesse di pubblicazioni di settori ben lontani dall'informatica come *Christian Science Monitor*.

All'inizio del nostro studio le società candidate erano 108, dopodiché il campo si restrinse a sedici finalisti, tra i quali c'erano la maggior parte dei principali produttori di database dell'epoca e tutti i tipi di database: di rete, gerarchici, relazionali e di altro tipo. Dopo l'ultima rigorosa tornata di domande, due dei principali produttori partecipanti chiesero che i risultati dello studio relativi ai loro prodotti non fossero mai pubblicati, mentre un venditore di una terza società lasciò il lavoro al termine di una delle sessioni. Sapevamo come porre domande difficili e scomode.

Oracle, nota allora con il nome di Relational Software, Inc., all'epoca aveva meno di 25 dipendenti e solo alcuni clienti di una certa importanza. Nondimeno, al termine dello studio annunciammo che Oracle era il vincitore. Dichiarammo che dal punto di vista tecnico Oracle era il miglior prodotto sul mercato e che il gruppo dirigenziale di RSI sembrava avere le capacità sufficienti a far progredire con successo la società. La nostra radicale proclamazione venne effettuata in un momento in cui alcuni non conoscevano nemmeno il significato del termine relazionale, mentre quei pochi che lo conoscevano non avevano molte cose positive da dire in proposito. Molti dirigenti di società informatiche criticarono apertamente le nostre conclusioni e predissero che Oracle e il database relazionale non sarebbero andati da nessuna parte.

Oracle oggi è la più importante società di produzione di database e la seconda società di software al mondo, mentre il database relazionale è ormai divenuto lo standard mondiale.

La Koch Systems Corporation, la società che possedevo e dirigivo a quell'epoca, fu la prima a diventare Valued Added Reseller di Oracle (ossia rivenditore di prodotti Oracle a valore aggiunto). Sviluppammo la prima importante applicazione relazionale commerciale al mondo, un sistema per la vendita e la gestione contabile dei titoli chiamato THESIS. Questo prodotto venne utilizzato da importanti banche e società per la gestione del loro portafoglio di investimenti. Persino IBM acquistò THESIS e consentì l'installazione di Oracle presso le proprie sedi, nono-

stante le forti resistenze interne. Dopo tutto IBM era la principale società di database a quell'epoca, con IMS e DB2 come prodotti di punta.

Oracle intanto continuava a rifinire il suo giovane prodotto, a capire quali fossero i tipi di caratteristiche e funzionalità che l'avrebbero reso produttivo e utile per il mondo aziendale, e il nostro lavoro di sviluppo alla Koch Systems contribuì a questa messa a punto. Alcune caratteristiche di Oracle furono il risultato diretto di nostre richieste agli sviluppatori di Oracle, mentre il nostro esplicito appoggio a una maggiore inclinazione verso l'utente finale nello sviluppo delle applicazioni e nelle convenzioni di denominazione ha influenzato un'intera generazione di programmatore che hanno imparato a lavorare con Oracle nella nostra società o che hanno letto gli articoli da noi pubblicati.

Questo stretto coinvolgimento con lo sviluppo e l'uso di Oracle ci portò a formarci una precoce e impareggiabile esperienza del prodotto e delle sue capacità. Dal momento che ho sempre amato condividere scoperte e conoscenze, per aiutare ad abbreviare il tempo di apprendimento necessario con le nuove tecnologie e idee e per evitare agli altri di commettere i miei stessi errori, decisi di trasformare ciò che avevo imparato in un libro.

*Oracle: La guida completa* venne concepito nel 1988 per raccogliere tutti i comandi e le tecniche fondamentali utilizzati nella linea di prodotti Oracle, nonché per offrire una solida guida alle modalità di sviluppo di applicazioni che utilizzano Oracle e SQL. La prima parte del libro era destinata senza distinzioni a sviluppatori e utenti finali, in modo che potessero condividere un linguaggio e una comprensione comuni durante il processo di sviluppo delle applicazioni: il concetto di sviluppatori e utenti finali che lavorano fianco a fianco era decisamente all'avanguardia quando venne ideato il libro.

Linda Allen, uno stimato agente letterario di San Francisco, mi presentò a Liz Fisher, all'epoca editor presso McGraw-Hill/Osborne. A Liz l'idea piacque molto. Venne stilato un contratto e l'uscita della prima edizione venne programmata per il 1989. Tuttavia un importante dirigente di McGraw-Hill, ora non più membro della società, venne a conoscenza del progetto e ne annullò immediatamente lo sviluppo, sostenendo che Oracle era un fuoco di paglia che non sarebbe andato da nessuna parte. Un anno più tardi, quando Oracle Corporation era nuovamente raddoppiata nelle dimensioni e il dirigente se n'era andato, il progetto venne ripreso e la prima edizione venne finalmente pubblicata nel 1990.

Quasi immediatamente questo divenne il libro più venduto nella sua categoria, e mantenne questa posizione per un decennio.

Nel luglio del 1990 venni assunto da Oracle per dirigere la sua Applications Division; venni nominato vicepresidente della società e guidai la divisione verso il successo mondiale (con l'aiuto di molti validi collaboratori). Mentre mi trovavo alla Oracle presentai McGraw-Hill/Osborne al gruppo dirigenziale di Oracle; da questo incontro, dopo l'opposizione di uno dei vicepresidenti della società (che ora non fa più parte di Oracle), che non capiva quale potesse essere il riscontro di questa idea, nacque Oracle Press, che attualmente è la più importante editrice al mondo di manuali di riferimento su Oracle.

Nel 1992 Bob Muller, uno dei primi sviluppatori di Koch Systems e Oracle, si assunse la responsabilità degli aggiornamenti tecnici del libro, in quanto gli impegni presso Oracle mi consentivano di occuparmi semplicemente della revisione editoriale delle modifiche. Questo lavoro portò a *Oracle7: La guida completa*. Questo fu il primo libro pubblicato da Bob, che da allora ha continuato a scrivere, dando vita a diversi libri molto noti sullo sviluppo e la progettazione di database.

Nel 1994 lasciai Oracle per coronare un desiderio che avevo da tempo, quello del ministero religioso a tempo pieno: attualmente sono il pastore della Church of the Resurrection (<http://www.resurrection.org>) di West Chicago, Illinois. Continuo a scrivere per diverse pubblicazioni come il *Wall Street Journal* e *Christianity Today* e di recente ho pubblicato un libro in Inghilterra.

ra, *The Country Parson's Advice to His Parishioner*, presso Monarch Books. Faccio anche parte del consiglio di amministrazione di Apropos, un'importante società che produce applicazioni per call center, ma non lavoro più allo sviluppo di applicazioni Oracle.

Sempre nel 1994, Kevin Loney, uno stimatissimo consulente indipendente su Oracle, autore di diversi testi (<http://www.kevinloney.com>), assunse l'incarico di riscrivere e aggiornare la terza edizione del libro, e da allora continua nell'opera. Egli ha contribuito con nuove importanti sezioni (come le guide e le sezioni su PL/SQL, Java e ORDBMS) e ha completamente aggiornato tutte le sezioni del libro alle nuove caratteristiche dei prodotti Oracle. Egli ha inoltre integrato molti commenti dei lettori nella struttura e nel contenuto del libro, trasformandolo, nella sua forma attuale, in prodotto che è il risultato del contributo di autori e lettori. Questi sforzi hanno consentito a *Oracle: La guida completa* di rimanere ai vertici nel suo campo e di continuare a essere la guida a Oracle più completa in assoluto, tuttora senza pari per estensione, contenuto e autorità. Sono un grande sostenitore di Kevin e sono davvero impressionato dalla sua conoscenza e precisione.

*Oracle: La guida completa* ormai è disponibile in otto lingue e si trova sulla scrivania di sviluppatori e utenti di prodotti Oracle di tutto il mondo. Non solo è stato il libro più venduto nella sua categoria (con due edizioni presenti in libreria è riuscito a essere contemporaneamente al primo e al quarto posto nelle vendite), ma si trova regolarmente nei primi cento tra *tutti* i libri venduti da Amazon.com. A un certo punto è stato al settimo posto tra i libri più venduti in Brasile! In termini di reputazione e successo non ha eguali sul mercato.

Come Oracle stessa, il libro è sopravvissuto e ha prosperato in barba alle ricorrenti predizioni di fallimento provenienti da molte parti. Forse questa breve storia può essere di incoraggiamento a chi è costretto ad affrontare opposizioni pur avendo una chiara visione di ciò che sarà necessario negli anni a venire.

Come disse Winston Churchill: “Mai arrendersi, mai arrendersi, mai arrendersi, per cose piccole, grandi o insignificanti che siano; mai arrendersi, se non alle convinzioni dettate dall'onore e dal buon senso”.

George Byron Koch  
GeorgeKoch@GeorgeKoch.com  
Wheaton, Illinois



# Introduzione

O racle è il database più diffuso al mondo. Funziona praticamente su tutti i tipi di computer e su tutte queste macchine il funzionamento è sostanzialmente identico; pertanto è sufficiente imparare a utilizzarlo su una piattaforma per poter passare alle altre senza problemi. Per questo utenti e sviluppatori di Oracle sono molto richiesti dalle aziende e risulta facile trasportare le proprie conoscenze e capacità in ambienti diversi.

La documentazione di Oracle è completa e assai voluminosa, tanto da comprendere diversi CD-ROM. *Oracle9i: La guida completa* è il primo libro che raccoglie tutte le principali definizioni, i comandi, le funzioni, le funzionalità e i prodotti legati a Oracle in un unico volume, una grande guida di riferimento che tutti gli utenti e gli sviluppatori dovrebbero tenere a portata di mano.

Le persone cui si rivolge questo libro possono essere distinte in tre categorie, descritte di seguito.

- *Utenti finali di Oracle* È possibile utilizzare Oracle per operazioni estremamente semplici come inserire dati ed eseguire report standard, ma in questo modo non se ne sfruttano le potenzialità; è come acquistare una potente macchina sportiva e poi trainarla con un cavallo. Con il materiale introduttivo fornito nelle prime due parti di questo libro, anche un utente finale quasi del tutto privo di esperienza nell'elaborazione dei dati può divenire un piccolo esperto di Oracle, in grado di generare report con indicazioni in italiano, guidare gli sviluppatori nella creazione di nuove funzionalità e migliorare la velocità e la precisione in un'azienda. Il linguaggio di questo libro è semplice e chiaro, privo di termini gergali; le conoscenze di computer o database richieste come presupposto sono assai ridotte. Il libro guida i principianti a diventare degli esperti con una struttura semplice da seguire e numerosi esempi concreti.
- *Sviluppatori che si avvicinano a Oracle per la prima volta* Con tutti i volumi di cui è dotata la documentazione di ORACLE, trovare un comando o un concetto importante può rivelarsi un'operazione molto lunga. Questo libro cerca di fornire un metodo meglio organizzato e più efficiente di apprendere i fondamenti del prodotto. Lo sviluppatore che non conosce Oracle viene condotto in una rapida panoramica sui concetti di base, accompagnato nelle aree più complesse, informato di possibili incomprensioni circa il prodotto e lo sviluppo di database relazionali, guidato con principi chiari per lo sviluppo di applicazioni.

- *Sviluppatori esperti in ORACLE* Come per tutti i prodotti di ampio respiro e molto sofisticati, esistono importanti aspetti riguardo ai quali vi è poca o nessuna documentazione. La conoscenza giunge con l'esperienza, ma spesso non viene trasferita agli altri. Questo libro approfondisce molti di questi argomenti, come la precedenza negli operatori UNION, INTERSECTION e MINUS, l'ereditarietà e CONNECT BY, l'eliminazione di NOT IN con un join esterno, l'utilizzo di interMedia, l'implementazione delle opzioni relazionali a oggetti e Java e molti altri. Nel testo sono inoltre chiariti molti aspetti che spesso sono oggetto di confusione e suggeriti rigorosi principi guida per le convenzioni di denominazione, le tecniche di sviluppo delle applicazioni e i problemi di progettazione e prestazioni.

## Struttura del libro

Il libro è suddiviso in sei parti e contiene un CD-ROM allegato.

La Parte prima è un'introduzione ai concetti fondamentali dei database. Si tratta di argomenti indispensabili per qualsiasi utente di Oracle, principiante o esperto, dai semplici addetti all'inserimento di dati ai DBA. Viene descritta la terminologia di base che utenti e sviluppatori possono utilizzare per condividere concetti in modo coerente e intelligente, al fine di garantire il successo di qualsiasi lavoro di sviluppo. Questa parte introduttiva si rivolge a sviluppatori e utenti finali di ORACLE, esamina i concetti di base e la terminologia dei database relazionali ed evidenzia i pericoli, gli errori classici e le opportunità offerte dalle applicazioni di database relazionali.

La Parte seconda spiega la teoria e le tecniche dei sistemi di database relazionali con le relative applicazioni, tra cui SQL (Structured Query Language) e SQLPLUS. Questa parte inizia con un numero relativamente limitato di presupposti circa la conoscenza delle tecniche di elaborazione dei dati da parte del lettore e prosegue passo per passo fino a raggiungere argomenti molto avanzati e tecniche complesse. Il testo è scritto ponendo molta attenzione all'uso di un italiano chiaro e scorrevole, con esempi unici e interessanti ed evitando l'uso di termini gergali o indefiniti. Questa parte è dedicata principalmente a sviluppatori e utenti finali che si accostano a Oracle per la prima volta, o che necessitano di un rapido ripasso di alcune caratteristiche di base. Vengono esaminate passo per passo le funzionalità di base di SQL e dello strumento per query interattive di Oracle, SQLPLUS. Una volta completata questa parte, il lettore avrà una completa conoscenza di tutte le parole chiave, le funzioni e gli operatori di SQL e sarà in grado di produrre report complessi, creare tabelle, inserire, aggiornare ed eliminare dati da un database Oracle.

Negli ultimi capitoli della Parte seconda sono presentati alcuni metodi avanzati per l'uso di SQLPLUS, l'interfaccia a riga di comando di Oracle, e descrizioni approfondite delle nuove e potenti caratteristiche di Oracle stesso. Questi argomenti sono utili per sviluppatori che hanno una certa familiarità con Oracle, soprattutto coloro che hanno utilizzato le versioni precedenti ma si sono resi conto di avere delle lacune. In alcuni casi si tratta di tecniche mai documentate o addirittura ritenute impossibili da realizzare. I suggerimenti e le tecniche avanzate trattate in questa parte mostrano come utilizzare ORACLE in modi potenti e creativi; tra gli altri argomenti sono trattati i modi per sfruttare le caratteristiche dei database distribuiti, il caricamento di file di dati e l'esecuzione di ricerche avanzate nel testo. Sono inoltre trattate le funzionalità più recenti, come le tabelle esterne, le query flashback, i nuovi tipi di dati e le nuove funzioni.

La parte terza offre una trattazione di PL/SQL; tra gli argomenti trattati troviamo un'analisi delle strutture PL/SQL, nonché dei trigger, delle stored procedure e dei package.

La Parte quarta è dedicata a un'ampia trattazione delle funzionalità orientate agli oggetti, come i tipi di dati astratti, i metodi, le viste oggetto, le tabelle oggetto, le tabelle annidate, gli array variabili e i LOB (Large Object).

La Parte quinta contiene invece una trattazione delle funzionalità Java presenti nel database Oracle; questa parte include un'introduzione generale alla sintassi di Java, oltre a capitoli su JDBC e SQLJ e sulle stored procedure Java.

La Parte sesta contiene diverse guide: al dizionario dati, all'ottimizzatore del database, a Oracle9i Application Server, all'amministrazione dei database e all'implementazione di XML di Oracle. Queste guide contengono un'introduzione generale alle aree che gli sviluppatori potrebbero aver bisogno di utilizzare nello sviluppo e nell'amministrazione delle loro applicazioni.

L'ultimo capitolo della Parte sesta contiene il riferimento completo per il server Oracle, che potrebbe costituire di per sé un intero libro. Le pagine introduttive sono molto utili per comprendere meglio il testo. Questa parte contiene i riferimenti relativi alla maggior parte dei comandi di Oracle, delle parole chiave, ai prodotti, alle funzionalità e alle funzioni, con ampi riferimenti incrociati tra gli argomenti. Questa guida di riferimento è stata studiata per essere consultata sia dagli utenti sia dagli sviluppatori di Oracle, ma richiede una certa familiarità con i prodotti. Per sfruttare al meglio la guida è consigliabile leggere le prime pagine introduttive, che spiegano in dettaglio gli argomenti trattati e mostrano come leggere il testo.

Il CD allegato al libro contiene il testo in versione elettronica, in modo che il lettore possa portarlo con sé sul proprio computer, mentre la versione cartacea rimane nel proprio ufficio oppure a casa. Il CD contiene inoltre le istruzioni per la creazione delle tabelle utilizzate nel libro, insieme con quelle per l'inserimento dei dati. Per chiunque stia imparando Oracle, la disponibilità di queste tabelle nel proprio ID, o in uno di esercizio, facilita notevolmente la comprensione degli esempi.

## Convenzioni di stile

Fatta eccezione per i controlli di uguaglianza (come Citta=‘CHICAGO’), ORACLE ignora la distinzione tra lettere maiuscole e minuscole. Nei listati di comandi, funzioni e della loro sintassi forniti nella guida di riferimento alfabetica del Capitolo 42 questo libro segue lo stile della documentazione di Oracle, secondo il quale tutti i comandi SQL sono posti in maiuscolo e tutte le variabili in minuscolo corsivo.

Tuttavia, la maggior parte degli utenti e degli sviluppatori di Oracle non utilizza le lettere maiuscole per i comandi SQL, perché è troppo faticoso e comunque per Oracle non conta nulla. Per questo motivo, nel libro si è scelto uno stile leggermente diverso per quanto riguarda gli esempi (e non nei comandi formali e nelle sintassi, come si è detto in precedenza), principalmente per migliorare la leggibilità. Le convenzioni di stile adottate sono descritte di seguito.

- Il corsivo e il grassetto non sono utilizzati nei listati di esempio.
- I termini select, from, where, order by, having e group by sono indicati in minuscolo e con un tipo di carattere diverso da quello del corpo del testo.
- I comandi di SQLPLUS sono riportati in minuscolo e con un tipo di carattere diverso da quello del corpo del testo; per esempio, column, set, save, ttitle e così via.
- Operatori e funzioni SQL, come IN, BETWEEN, UPPER, SOUNDEX e così via sono riportati in maiuscolo e con un tipo di carattere diverso da quello del corpo del testo.
- Per le colonne si utilizzano nomi in maiuscolo e minuscolo, come per Caratteristica, EstOvest, Longitudine e così via.
- I nomi delle tabelle, come GIORNALE, CLIMA, DISLOCAZIONE e così via, sono riportati in maiuscolo.



Parte prima

**CONCETTI  
FONDAMENTALI  
DEI DATABASE**



## Capitolo 1

# Condividere conoscenze e successo

- 1.1 L'approccio cooperativo
- 1.2 Tutti hanno dei "dati"
- 1.3 Il linguaggio familiare di Oracle
- 1.4 Alcuni esempi comuni

Per poter creare e utilizzare rapidamente ed efficacemente un'applicazione Oracle9i, gli utenti e gli sviluppatori devono condividere un linguaggio comune e avere una conoscenza approfondita sia del compito specifico per cui viene utilizzata l'applicazione, sia degli strumenti di Oracle.

Si tratta di un nuovo approccio allo sviluppo dei programmi software. In passato, gli analisti di sistemi studiavano le esigenze del mondo del lavoro e progettavano un applicazione che soddisfacesse tali esigenze. L'utente veniva coinvolto solo nella fase di descrizione del lavoro e talvolta nella revisione della funzionalità dell'applicazione progettata.

Con i nuovi strumenti e le varie modalità di approccio disponibili, e soprattutto con Oracle, è possibile sviluppare applicazioni più adeguate alle aspettative e alle abitudini del mondo del lavoro, a condizione però che si instauri un linguaggio comune.

Lo scopo specifico di questo libro è promuovere questa condivisione di conoscenze e fornire agli utenti e agli sviluppatori i mezzi per sviluppare appieno le potenzialità di Oracle. L'utente finale conoscerà dettagli del lavoro che lo sviluppatore non è in grado di comprendere. Lo sviluppatore apprenderà funzioni e caratteristiche interne di Oracle e dell'ambiente informatico che sarebbero tecnicamente troppo complesse per l'utente finale. Tuttavia, questi ambiti di conoscenza esclusiva hanno portata minore rispetto a tutto ciò che utenti e sviluppatori possono condividere utilizzando Oracle; si tratta di un'opportunità davvero interessante.

Non è un segreto che il "mondo del lavoro" e il "mondo degli informatici" sono stati in conflitto per molti anni. Differenze culturali e ambiti di conoscenza diversi, interessi e obiettivi professionali differenti, oltre al senso di estraneità che la semplice separazione fisica talvolta produce, sono alcuni dei motivi alla base di questa situazione. Per essere giusti, questa sorta di sindrome non è tipica dell'informatizzazione; la stessa situazione si verifica anche tra persone che si occupano di contabilità, gestione del personale o direzione generale, poiché i componenti di ogni gruppo tendono a raggrupparsi e a separarsi dagli altri gruppi collocandosi fisicamente su un piano, un edificio o una città diversa. Le relazioni tra individui di gruppi differenti diventano formali e tese. Si instaurano delle barriere e delle procedure artificiali che prendono le mosse da questo isolazionismo, contribuendo ad aggravare la "sindrome".

Va bene, si potrebbe pensare, tutto questo può essere interessante per i sociologi, ma che cosa ha a che fare con Oracle?

*Il fatto è che Oracle non si fonda su un linguaggio arcano comprensibile soltanto per gli specialisti, ma modifica la natura del rapporto tra utenti e sviluppatori.* Chiunque può comprenderlo, chiunque può utilizzarlo. Le informazioni che prima erano intrappolate nei sistemi informatici finché qualcuno non creava un report di presentazione sono ora accessibili, istantaneamente.

mente, a chiunque, semplicemente inserendo una query in lingua inglese. In questo modo cambiano le regole del gioco.

Utilizzando Oracle, la comprensione tra i due campi è migliorata radicalmente, è aumentata la conoscenza reciproca e anche le relazioni tra i due gruppi hanno cominciato a normalizzarsi. Di conseguenza, sono state ottenute applicazioni e risultati finali di qualità superiore.

Fin dalla prima versione, Oracle è stato impostato su un modello relazionale di facile comprensione (spiegato tra breve); in questo modo i non esperti possono comprendere prontamente che cosa fa Oracle e come lo fa. Ciò lo ha reso di facile approccio.

Inoltre Oracle è stato progettato per essere eseguito praticamente nello stesso modo su qualsiasi tipo di computer. Qualunque fosse il produttore dell'attrezzatura utilizzata dall'utente, Oracle funzionava. Tutte queste caratteristiche contribuirono direttamente al grande successo del prodotto e della società.

In un mercato popolato da società che producono hardware “proprietario”, sistemi operativi “proprietari”, database “proprietari” e applicazioni “proprietarie”, Oracle consente agli utenti che lo utilizzano per lavoro e ai settori di sviluppo dei sistemi un nuovo controllo sulle loro vite e sul loro futuro, senza legami con il database di un unico venditore di hardware. Oracle può essere eseguito su qualsiasi tipo di computer. Si tratta di una rivoluzione fondamentale per il mondo del lavoro e per lo sviluppo di applicazioni, con conseguenze che si proietteranno molto lontano nel futuro.

Alcuni individui non lo accettano né comprendono, né capiscono l'importanza vitale del fatto che le vecchie e artificiali barriere tra “utenti” e “sistemi” continuino a crollare, ma l'avvento dello sviluppo cooperativo avrà influenza profonda sulle applicazioni e sulla loro utilità.

Tuttavia, molti sviluppatori di applicazioni sono caduti in una trappola: proseguire con metodi inutili ereditati dalla progettazione di sistemi della generazione precedente. C'è molto da disimparare. Molte delle tecniche (e dei limiti) che erano indispensabili in precedenza non sono soltanto inutili nella progettazione con Oracle, ma del tutto controproducenti. Nell'affrontare l'apprendimento di Oracle ci si deve liberare dal peso di queste abitudini e di questi approcci datati: ora sono disponibili nuove e stimolanti possibilità.

L'intento fondamentale di questo testo è spiegare Oracle in maniera semplice e chiara, in termini comprensibili e condivisibili sia dagli utenti, sia dai tecnici. Per quanto concerne invece le modalità di progettazione e gestione dimate o inadeguate, verrà proposto il modo migliore per sostituirle.

## 1.1 L'approccio cooperativo

Oracle è un *database relazionale a oggetti*. Un database relazionale è un sistema estremamente semplice per considerare e gestire i dati utilizzati in un lavoro. Non è niente di più di un insieme di tabelle di dati. Le tabelle sono molto presenti nella vita quotidiana: condizioni climatiche, grafici sull'andamento della borsa, risultati di avvenimenti sportivi sono tutte tabelle, dotate di intestazioni di colonna e righe con le informazioni presentate in maniera semplice. L'approccio relazionale può essere sofisticato e sufficientemente espressivo anche per il lavoro più complesso. Un database relazionale a oggetti presenta tutte le caratteristiche di un database relazionale, ma consente anche di sviluppare concetti e funzioni orientati agli oggetti.

Purtroppo, proprio coloro che possono trarre più utilità da un database relazionale, ovvero gli utenti che se ne servono per lavoro, in genere ne sanno di meno. Gli sviluppatori di applicazioni, che costruiscono i sistemi che questi utenti utilizzano nella loro attività, spesso pensano che i concetti che stanno alla base del software relazionale siano troppo difficili per poter essere

spiegati in termini semplici. È necessario un linguaggio comune perché l'approccio cooperativo funzioni.

Nelle prime due parti di questo testo viene illustrato, con termini di comprensibilità immediata, che cos'è esattamente un database relazionale e come utilizzarlo efficacemente nella propria attività. A prima vista, potrebbe sembrare che questa trattazione sia esclusivamente a beneficio degli "utenti", quindi un tecnico esperto di applicazioni relazionali potrebbe avere la tendenza a saltare questi primi capitoli per utilizzare il libro semplicemente come testo elementare di consultazione su Oracle. Meglio resistere a questa tentazione! Anche se molto del materiale può apparire come una panoramica di base, per i progettisti di applicazioni si tratta di un'opportunità per acquisire un terminologia chiara, coerente e gestibile di cui servirsi per conversare con gli utenti riguardo alle loro esigenze e alle modalità per soddisfarle velocemente. Per i lettori di questo tipo, questa trattazione può anche essere utile per abbandonare alcune abitudini inutili e probabilmente inconsce tipiche dei tecnici, molte delle quali vengono svelate durante la presentazione dell'approccio relazionale. È importante comprendere che persino il potenziale di Oracle può essere limitato considerevolmente da metodi di progettazione adatti soltanto allo sviluppo di software non relazionale.

Per gli utenti finali, la comprensione dei concetti fondamentali dei database relazionali a oggetti consente di esprimere meglio le proprie esigenze agli sviluppatori di applicazioni e di capire come possano essere soddisfatte. Mediamente un operatore può passare da principiante a esperto in breve tempo. Oracle offre la capacità di raccogliere e utilizzare informazioni, di avere il controllo immediato dei report e dei dati e una visione chiara di come funziona l'applicazione. L'utente di Oracle può gestire un'applicazione o una query in maniera avanzata e sapere se non sta utilizzando tutta la flessibilità e il potenziale disponibile.

L'utente finale ha anche la possibilità di liberare i programmati dal loro compito meno gradevole: compilare nuovi report. Nelle grandi aziende, ben il 95 per cento di tutto il cumulo di lavoro arretrato è composto da richieste di nuovi report. Dato che l'utente è in grado di compilarli autonomamente, in termini di minuti e non di mesi, sarà gratificante averne la responsabilità.

## 1.2 **Tutti hanno dei "dati"**

In una biblioteca vengono conservati elenchi di iscritti, libri e registrazioni di prestiti. Il proprietario di una collezione di figurine di calcio registra nomi, date, medie dei giocatori e valore delle figurine. In qualsiasi attività commerciale devono essere conservate alcune informazioni specifiche su clienti, prodotti, prezzi. Tutte queste informazioni sono definite *dati*.

I filosofi dell'informazione amano dire che i dati rimangono semplici dati finché non vengono organizzati in maniera significativa, diventando solo a quel punto "informazioni". Se questa tesi è corretta, Oracle è anche un mezzo per trasformare facilmente i dati in informazioni. Con Oracle i dati vengono selezionati e manipolati per portare alla luce le conoscenze che nascondono, come totali, tendenze di mercato o altre relazioni che fino a quel momento non sono palesi. I lettori impareranno come fare queste scoperte. Il concetto fondamentale in questo contesto è rappresentato dai dati e dalle tre operazioni che possono essere effettuate con essi: acquisirli, memorizzarli e richiamarli.

Una volta chiarite le nozioni di base, si può procedere effettuando conteggi, spostando i dati da una collocazione all'altra e modificarli. Ciò viene definito come *elaborazione* e, fondamentalmente, coinvolge le tre fasi con cui vengono organizzate le informazioni.

Tutte queste operazioni potrebbero essere effettuate con carta e matita, ma con l'aumentare del volume dei dati tendenzialmente cambiano anche gli strumenti. È possibile utilizzare un

classificatore, calcolatrici, matite e carta. A un certo punto diventa sensato passare ai computer, anche se i compiti da eseguire rimangono gli stessi.

Un *sistema di gestione di database relazionale* (o RDBMS, acronimo di Relational DataBase Management System) come Oracle consente di svolgere questi compiti in maniera comprensibile e ragionevolmente semplice. In pratica con Oracle è possibile svolgere tre compiti:

- inserire dati;
- mantenere i dati in memoria;
- reperire i dati e gestirli.

Nella Figura 1.1 viene illustrata la semplicità di tutto ciò.

Oracle supporta questo approccio in tre fasi e fornisce strumenti intelligenti che consentono un notevole livello qualitativo nelle modalità con cui i dati vengono reperiti, visualizzati, modificati e inseriti, mantenuti al sicuro ed estrapolati per manipolarli e costruire report sulla base delle informazioni che contengono.

Un *sistema di gestione di database relazionale a oggetti* (ORDBMS) estende le possibilità di un RDBMS per supportare concetti orientati agli oggetti. Oracle può essere utilizzato come RDBMS per sfruttare le sue caratteristiche orientate agli oggetti.

### 1.3 Il linguaggio familiare di Oracle

Le informazioni memorizzate in Oracle vengono gestite in forma di tabelle analoghe a quelle delle previsioni meteorologiche dei quotidiani, come l'esempio illustrato nella Figura 1.2.

In questa tabella sono presenti quattro colonne verticali: Città, Temperatura, Umidità e Condizione. C'è anche una riga per ogni città da Atene a Sidney. Infine, la tabella ha un nome: CLIMA.

Si tratta delle tre caratteristiche principali della maggior parte delle tabelle stampate: *colonne, righe e nome*. Lo stesso accade per un database relazionale. I termini e i concetti rappresentati sono comprensibili da tutti, perché le parole utilizzate per descrivere i componenti di una

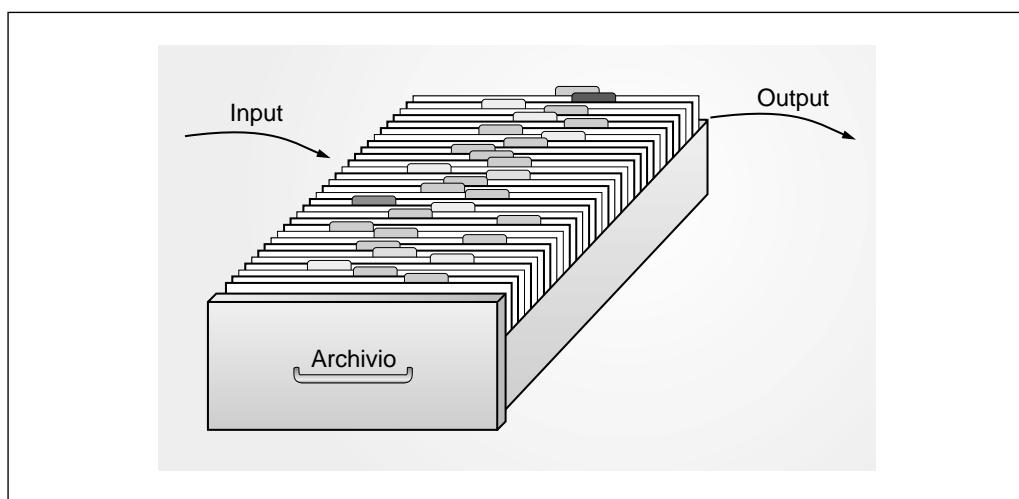


Figura 1.1 Gestione dei dati in Oracle.

CLIMA			
Città	Temperatura	Umidità	Condizione
ATENE.....	36	89	SOLE
CHICAGO.....	19	88	PIOGGIA
LIMA.....	7	79	PIOGGIA
MANCHESTER....	19	98	NEBBIA
PARIGI.....	27	62	NUVOLOSO
SPARTA.....	23	63	NUVOLOSO
SYDNEY.....	-5	12	NEVE

Figura 1.2 Una tabella di dati meteorologici tratta da un giornale.

tabella in un database di Oracle sono le stesse utilizzate nelle conversazioni quotidiane. Non si tratta di termini dal significato particolare, insolito o esoterico: hanno esattamente il significato che suggeriscono.

## Tabelle di informazioni

Oracle memorizza le informazioni in tabelle, delle quali è possibile vedere un esempio nella Figura 1.3. In ciascuna di queste tabelle sono presenti una o più colonne. Le intestazioni delle colonne come Città, Temperatura, Umidità e Condizione visibili nella Figura 1.3 sono utilizzate per descrivere il tipo di informazione contenuta nella colonna. Le informazioni vengono memorizzate riga dopo riga (città dopo città). Ogni singolo insieme di dati, come la temperatura, l'umidità e la condizione per la città di Manchester, occupa la sua colonna specifica.

Con Oracle viene evitata la terminologia specializzata, accademica, al fine di rendere il prodotto di più facile approccio. Nei documenti di ricerca sulla teoria relazionale, una colonna può essere definita “attributo”, una riga “tupla” e una tabella può essere definita come “entità”. Per l’utente finale, tuttavia, questi termini possono generare confusione. Inoltre, si tratta di una ridenominazione non necessaria di elementi per i quali esistono già termini generalmente com-

Nome tabella			
Città	Temperatura	Umidità	Condizione
ATENE	36	89	SOLE
CHICAGO	19	88	PIOGGIA
LIMA	7	79	PIOGGIA
MANCHESTER	19	98	NEBBIA
PARIGI	27	62	NUVOLOSO
SPARTA	23	63	NUVOLOSO
SYDNEY	-5	12	NEVE

Figura 1.3 Una tabella di dati meteorologici in Oracle.

presi nel linguaggio quotidiano. Con Oracle si utilizza questo linguaggio quotidiano e anche i tecnici possono adeguarsi. È indispensabile prendere coscienza del muro di sfiducia e incomprendensione che viene prodotto dall'uso di gergo tecnico non necessario. Come Oracle, quindi, anche questo testo si basa su "tabelle", "colonne" e "righe".

## SQL

Oracle è stata la prima azienda a presentare un prodotto che utilizzava il *linguaggio di interrogazione strutturato SQL* (Structured Query Language), basato sulla lingua inglese. Questo linguaggio consente agli utenti finali di estrarre le informazioni in modo autonomo, senza rivolgersi ai sistemisti per ogni piccolo report.

Il linguaggio di interrogazione di ORACLE è dotato di struttura, proprio come la lingua inglese o qualunque altra lingua. Ci sono regole grammaticali e sintattiche, ma sono fondamentalmente le stesse di un discorso corretto e risultano quindi di immediata comprensione.

Come verrà illustrato tra breve, SQL è uno strumento sorprendentemente potente, il cui utilizzo non richiede nessuna esperienza di programmazione.

Ecco un esempio di possibile impiego: se qualcuno chiede di selezionare dalla tabella CLIMA la città con umidità pari a 89, si può facilmente rispondere "Atene". Se venisse chiesto di selezionare le città con temperatura pari a 19 gradi, la risposta sarebbe "Chicago e Manchester".

Anche Oracle è in grado di rispondere alle stesse domande, quasi con la stessa facilità di un operatore e in risposta a una query semplice, molto simile alle domande poste poc'anzi. Le parole chiave utilizzate in una query per ORACLE sono select, from, where e order by. Si tratta di spunti che consentono a Oracle di comprendere la domanda e presentare la risposta corretta.

### Una semplice query di Oracle

In ORACLE, con una tabella CLIMA, la prima query che si potrebbe impostare (con un punto e virgola che indichi a Oracle che deve eseguire il comando) sarebbe semplicemente:

```
select Citta from CLIMA where Umidita = 89 ;
```

La risposta sarebbe:

```
Citta  
-----  
Atene
```

La seconda query potrebbe essere:

```
select Citta from CLIMA where Temperatura = 19 ;
```

La risposta a questa query sarebbe:

```
Citta  
-----  
Manchester  
Chicago
```

Come si può vedere, ciascuna di queste query usa le parole chiave **select**, **from** e **where**. Che dire di **order by**? Si supponga di voler visualizzare tutte le città in ordine di temperatura. Occorrerebbe semplicemente digitare:

```
select Citta, Temperatura from CLIMA
order by Temperatura ;
```

e la risposta immediata sarebbe:

Citta	Temperatura
SYDNEY	-5
LIMA	7
MANCHESTER	19
CHICAGO	19
SPARTA	23
PARIGI	27
ATENE	36

Oracle ha rapidamente riordinato la tabella per temperatura (in questa tabella sono elencate per prime le temperature più basse; più avanti verrà spiegato come specificare l'ordinamento).

Con gli strumenti di query di Oracle è possibile impostare molte altre domande, tuttavia gli esempi riportati dimostrano quanto sia facile ricavare informazioni da un database Oracle nella forma più corrispondente alle esigenze dell'utente. Si possono costruire richieste complesse partendo da informazioni semplici, ma il metodo utilizzato rimane sempre facilmente comprensibile. Per esempio, si possono combinare i parametri **where** e **order by**, entrambi elementi semplici, per chiedere al programma di selezionare le città in cui la temperatura è superiore a 26 gradi e di visualizzarle in ordine di temperatura crescente. In questo caso occorre digitare:

```
select Citta, Temperatura from CLIMA
where Temperatura > 26
order by Temperatura ;
```

e la risposta immediata sarebbe:

Citta	Temperatura
PARIGI	27
ATENE	36

Oppure si può impostare una richiesta ancora più specifica, chiedendo quali sono le città in cui la temperatura è superiore a 26 gradi e l'umidità inferiore a 70:

```
select Citta, Temperatura, Umidita from CLIMA
where Temperatura > 26
and Umidita < 70
order by Temperatura ;
```

e la risposta immediata sarebbe:

Citta	Temperatura	Umidita
PARIGI	27	62

## Che cosa significa relazionale

Occorre osservare che nella tabella CLIMA sono elencate città di vari Paesi e che per alcuni Paesi sono presenti diverse città. Si supponga di voler conoscere in quale Paese è collocata una data città. Si potrebbe creare una tabella DISLOCAZIONE separata per le città e i rispettivi paesi, come illustrato nella Figura 1.4.

Per ciascuna città nella tabella CLIMA è possibile semplicemente dare un'occhiata alla tabella DISLOCAZIONE, trovare il nome nella colonna Città, scorrere la colonna Paese nella stessa riga e verificare il nome del paese.

Si tratta di due tabelle completamente separate e indipendenti; ciascuna contiene determinate informazioni suddivise in colonne e righe e ha un elemento significativo in comune: la colonna Città. A ogni nome di città nella tabella CLIMA corrisponde un nome identico nella tabella DISLOCAZIONE.

Per esempio, si supponga di voler conoscere la temperatura, l'umidità e le condizioni climatiche di una città australiana. È sufficiente osservare le due tabelle per giungere alla risposta.

CLIMA			
Città	Temperatura	Umidità	Condizione
ATENE	36	89	SOLE
CHICAGO	19	88	PIOGGIA
LIMA	7	79	PIOGGIA
MANCHESTER	19	98	NEBBIA
PARIGI	27	62	NUVOLOSO
SPARTA	23	63	NUVOLOSO
SYDNEY	-5	12	NEVE

DISLOCAZIONE	
Città	Paese
ATENE	GRECIA
CHICAGO	STATI UNITI
CONAKRY	GUINEA
LIMA	PERU
MADRAS	INDIA
MADRID	SPAGNA
MANCHESTER	INGHILTERRA
MOSCA	RUSSIA
PARIGI	FRANCIA
ROMA	ITALIA
SHENYANG	CINA
SPARTA	GRECIA
SYDNEY	AUSTRALIA
TOKIO	GIAPPONE

Figura 1.4 Le tabelle CLIMA e DISLOCAZIONE.

Come si risolve questo quesito? È presente una sola voce AUSTRALIA, nella colonna Paese, nella tabella DISLOCAZIONE. Accanto a essa, nella colonna Citta della stessa riga è visualizzato il nome della città, SYDNEY. Si cerca questo nome nella colonna Citta della tabella CLIMA e, una volta individuato, ci si sposta lungo la riga per trovare i valori di Temperatura, Umidità e Condizione: -5, 12 e NEVE.

Anche se le tabelle sono indipendenti, si può facilmente constatare che risultano correlate: un nome di città in una tabella è correlato a un nome di città nell'altra (Figura 1.5). Il termine *database relazionale* fa riferimento a questo tipo di relazione.

Questa è la nozione fondamentale di database relazionale (talvolta definito *modello relazionale*). I dati vengono organizzati in tabelle, composte da colonne, righe e nomi. Le tabelle possono essere correlate tra loro se hanno una colonna con un tipo di informazione comune.

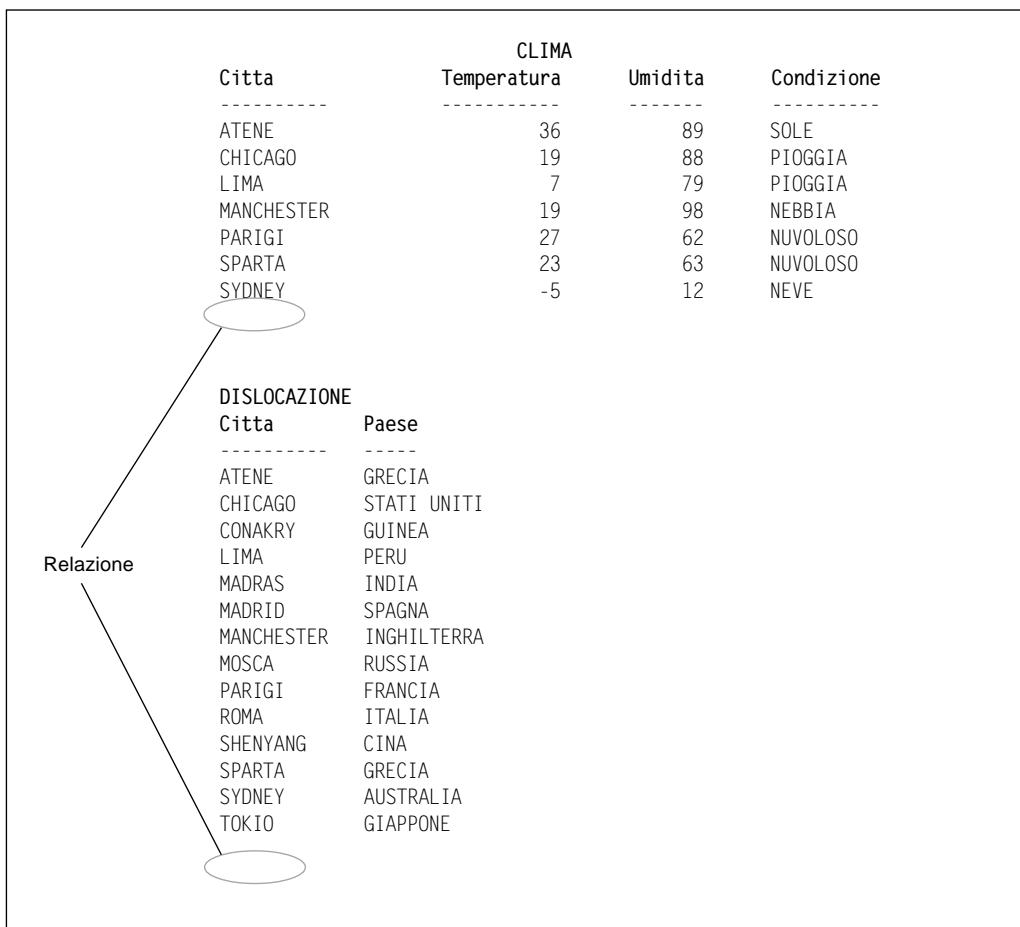


Figura 1.5 Le relazioni tra le tabelle CLIMA e DISLOCAZIONE.

## 1.4 Alcuni esempi comuni

Una volta compreso il concetto fondamentale di database relazionale, si vedono tabelle, righe e colonne ovunque. Naturalmente tabelle, righe e colonne erano presenti anche prima, ma cambia il modo di considerarle. Molte delle tabelle che si osservano più comunemente potrebbero essere inserite in Oracle, così da poterle utilizzare per rispondere velocemente a domande per le quali sarebbe necessario molto tempo in qualsiasi altro modo.

Nella Figura 1.6 è rappresentata una tipica tabella sull'andamento della borsa. Si tratta di una piccola parte di un elenco alfabetico molto denso che comprende molte colonne ravvicinate distribuite su diverse pagine in un giornale. Per quale azienda è stato scambiato il numero maggiore di azioni? Quale ha avuto la variazione percentuale più elevata, in positivo o in negativo? Per ottenere queste risposte con Oracle è sufficiente impostare delle semplici query; i tempi di risposta sono molto più rapidi di quelli impiegati dovendo scorrere le colonne sulla pagina di un giornale.

Azienda	Chiusura ieri	Chiusura oggi	Azioni scambiate
Ad Specialty	31.75	31.75	18,333,876
Apple Cannery	33.75	36.50	25,787,229
AT Space	46.75	48.00	11,398,323
August Enterprises	15.00	15.00	12,221,711
Brandon Ellipsis	32.75	33.50	25,789,769
General Entropy	64.25	66.00	7,598,562
Geneva Rocketry	22.75	27.25	22,533,944
Hayward Antiseptic	104.25	106.00	3,358,561
IDK	95.00	95.25	9,443,523
India Cosmetics	30.75	30.75	8,134,878
Isaiah James Storage	13.25	13.75	22,112,171
KDK Airlines	80.00	85.25	7,481,566
Kentgen Biophysics	18.25	19.50	6,636,863
LaVay Cosmetics	21.50	22.00	3,341,542
Local Development	26.75	27.25	2,596,934
Maxtide	8.25	8.00	2,836,893
MBK Communications	43.25	41.00	10,022,980
Memory Graphics	15.50	14.25	4,557,992
Micro Token	77.00	76.50	25,205,667
Nancy Lee Features	13.50	14.25	14,222,692
Northern Boreal	26.75	28.00	1,348,323
Ockham Systems	21.50	22.00	7,052,990
Oscar Coal Drayage	87.00	88.50	25,798,992
Robert James Apparel	23.25	24.00	19,032,481
Soup Sensations	16.25	16.75	22,574,879
Wonder Labs	5.00	5.00	2,553,712

Figura 1.6 Una tabella di dati di borsa.

Argomento	Sezione	Pagina
Nascite	F	7
Bridge	B	2
Economia	E	1
Annunci	F	8
Fumetti	C	4
Salute	F	6
Editoriali	A	12
Vita moderna	B	1
Fiml	B	4
Notizie	A	1
Necrologi	F	6
Sport	D	1
Televisione	B	7
Meteo	C	2

**Figura 1.7** Una tabella basata sulle sezioni di un giornale.

Nella Figura 1.7 è riprodotto un indice di giornale. Che cosa si trova nella sezione F? Se si leggesse il giornale dall'inizio alla fine, in che ordine apparirebbero gli articoli? Anche in questo caso, per ottenere le risposte a queste domande con Oracle è sufficiente impostare delle semplici query. In questo libro viene illustrato come impostare tutte queste query e anche come costruire le tabelle in cui conservare le informazioni.

In tutto il libro gli esempi utilizzeranno dati e oggetti che si incontrano frequentemente sul lavoro e nella vita quotidiana. Non dovrebbe essere difficile trovare dati simili da usare per i propri esercizi. Nelle pagine a seguire si imparerà a inserire e recuperare dati con il ricorso a esempi basati su origini dati tratte dalla vita quotidiana.



## Capitolo 2

# I pericoli di un database relazionale

- 2.1 **È davvero facile come sembra?**
- 2.2 **Quali sono i rischi?**
- 2.3 **L'importanza della nuova visione**
- 2.4 **Codici, abbreviazioni e standard di denominazione**
- 2.5 **Come arginare la confusione**
- 2.6 **Maiuscole e minuscole nei nomi e nei dati**
- 2.7 **Normalizzazione dei nomi**
- 2.8 **L'importanza del fattore umano**
- 2.9 **Comprensione dei dati**
- 2.10 **Verso la normalizzazione dei nomi degli oggetti**
- 2.11 **Chiavi intelligenti e valori di colonna**
- 2.12 **Il decalogo**

Come accade per qualsiasi nuova tecnologia o per un nuovo avvenimento dall’incerto profilo, è importante considerare attentamente non solo i benefici e le opportunità che si presentano, ma anche i costi e i rischi. Se a ciò si aggiunge un database relazionale con una serie di strumenti potenti e facili da utilizzare, come Oracle, la possibilità di essere guidati verso il disastro proprio a causa di questa semplicità diventa reale. Se si utilizzano anche funzioni orientate agli oggetti e al Web, il pericolo aumenta. In questo capitolo vengono discussi alcuni dei pericoli che progettisti e utenti dovrebbero considerare.

### 2.1 È davvero facile come sembra?

Secondo i fornitori di database, lo sviluppo di un’applicazione con il ricorso a un database relazionale e agli strumenti correlati di “quarta generazione” risulterà addirittura venti volte più rapido rispetto all’adozione della progettazione di sistema tradizionale. È molto facile: in ultima analisi, i programmati e gli analisti di sistema non risulteranno più indispensabili poiché gli utenti finali controlleranno i loro destini.

I critici dell’approccio relazionale tuttavia ritengono che i sistemi relazionali siano per natura più lenti degli altri, che gli utenti cui viene dato il controllo sulla formulazione di query e report avranno la meglio sui computer e che le società perderanno il loro ruolo, anche economicamente, se non si adotterà un approccio più tradizionale. La stampa riporta resoconti su applicazioni di grande portata che non hanno funzionato una volta messe in produzione.

Qual è allora la verità? Che le regole del gioco sono cambiate. Lo sviluppo di quarta generazione presuppone richieste molto diverse nei confronti delle società e della gestione rispetto ai modelli di sviluppo più tradizionali. Si tratta di esigenze e di rischi totalmente nuovi e niente affatto ovvi; una volta identificati e compresi, il pericolo non è superiore, ma anzi, probabilmente, molto inferiore rispetto allo sviluppo tradizionale.

## 2.2 Quali sono i rischi?

Il rischio principale è rappresentato dal fatto che lo sviluppo di applicazioni che utilizzano database relazionali è davvero facile come sembra. Comprendere le tabelle, le colonne e le righe non è difficile. La relazione tra due tabelle risulta concettualmente semplice. Anche la *normalizzazione*, il processo di analisi delle relazioni naturali o “normali” tra i vari elementi dei dati di una società, è piuttosto semplice da apprendere.

Purtroppo, questa semplicità spesso produce “esperti” pieni di sicurezza ma con poca esperienza nella creazione di applicazioni reali ed efficaci. Nel caso di un piccolo database di marketing, o di un’applicazione per l’inventario di casa, ciò non ha molta importanza; gli errori commessi vengono a galla con il tempo, le lezioni vengono imparate e la volta successiva vengono evitati. In un’applicazione importante, tuttavia, questa è la formula sicura per ottenere un disastro. Spesso è proprio la mancanza di esperienza ciò che sta dietro ai racconti riportati sulla stampa riguardo a fallimenti di grandi progetti.

I metodi di sviluppo più datati sono generalmente più lenti, fondamentalmente in quanto i loro compiti, codifica, compilazione, linking e collaudo, vengono effettuati con un ritmo più lento. Il ciclo, soprattutto per un mainframe, è spesso così tedioso che i programmati dedicano molto tempo a “controllare a tavolino” il prodotto onde evitare il rallentamento di un altro ciclo completo nel caso si presenti un errore nel codice.

Gli strumenti di quarta generazione spingono i progettisti ad accelerare il processo produttivo. I cambiamenti possono essere apportati e implementati così velocemente che rimane poco spazio per le verifiche. L’eliminazione di tutti i controlli a tavolino complica ulteriormente il problema. Quando l’incentivo negativo (il ciclo completo) che aveva promosso la prassi dei controlli a tavolino scompare, questi ultimi scompaiono. L’inclinazione di molti sembra essere quella per cui, se l’applicazione non è perfetta, è possibile rimediare velocemente; se si verificano problemi con i dati, si possono risolvere con un rapido aggiornamento; se il prodotto non è abbastanza veloce, si può metterlo a punto mentre lo si utilizza: l’importante è portarlo a compimento prima della scadenza prefissata e dimostrare di che cosa si è capaci.

Questo problema viene peggiorato da un interessante fenomeno sociologico: molti degli sviluppatori di applicazioni relazionali sono neolaureati che hanno imparato la teoria e la progettazione relazionale o ad oggetti a scuola e sono pronti a lasciare la loro impronta. Gli sviluppatori più esperti, invece, non hanno imparato la nuova tecnologia: sono occupati a supportare e migliorare le tecnologie che conoscono, che stanno alla base dei sistemi informativi correnti delle loro aziende. Il risultato è che gli sviluppatori inesperti tendono a porre l’accento sui progetti relazionali, sono talvolta meno inclini a effettuare verifiche e sono meno sensibili alle conseguenze di un fallimento rispetto a coloro che hanno già sperimentato vari cicli completi di sviluppo di applicazioni.

Il ciclo di collaudo in un progetto importante per Oracle dovrebbe essere più lungo e più completo rispetto a quello di un progetto tradizionale. Questo vale anche quando vengono impostati controlli di progetto appropriati e anche se il progetto è guidato da dirigenti esperti, poiché ci sono meno procedure di verifica a tavolino e una sopravvalutazione innata. Con questi collaudi dev’essere controllata la correttezza delle schermate e dei report, del caricamento e dell’ag-

giornamento, dell'integrità e della coerenza dei dati e soprattutto dei volumi delle transazioni e della memorizzazione durante i periodi di massimo carico.

Proprio perché è davvero facile come sembra, lo sviluppo di applicazioni con gli strumenti di Oracle può essere sorprendentemente rapido. Tuttavia, si riduce automaticamente la quantità di collaudi svolti come normale procedura dello sviluppo, e collaudi e certificazioni di qualità pianificati devono essere volontariamente allungati per compensare questa mancanza. Ciò non è sempre previsto dai progettisti poco esperti di Oracle o degli strumenti di quarta generazione, tuttavia è opportuno tenerne conto nella pianificazione del progetto.

## 2.3 L'importanza della nuova visione

Molti attendono con ansia il giorno in cui si potrà semplicemente digitare interrogazioni in linguaggio corrente e ottenere la risposta sullo schermo del computer nel giro di pochi secondi.

Siamo molto più vicini a questo obiettivo di quanto pensi la maggior parte delle persone. Il fattore limitativo non è più la tecnologia, ma la rigidità del pensiero nella progettazione di applicazioni. Con Oracle è facile concepire sistemi basati sulla lingua naturale che siano comprensibili e di uso immediato da parte degli utenti non esperti. Il potenziale è questo, già disponibile con il database e gli strumenti di Oracle, ma soltanto pochi lo hanno compreso e sono in grado di sfruttarlo.

La chiarezza e la comprensibilità dovrebbero essere i tratti distintivi di qualsiasi applicazione Oracle. È possibile utilizzare la lingua corrente, al punto che gli utenti finali senza preparazione specifica di programmazione possono usare facilmente le applicazioni e ottenere le informazioni desiderate formulando una semplice query.

E come si fa a ottenere tutto ciò? Innanzitutto, uno degli obiettivi fondamentali durante la programmazione dev'essere quello di rendere l'applicazione facile da capire e semplice da utilizzare. Anche se si sbaglia, questa è la direzione da seguire, persino se significa utilizzare più tempo della CPU o spazio su disco. Il limite di questo tipo di approccio è quello di creare programmi eccessivamente complessi che risultino quasi impossibili da gestire e migliorare. Sarebbe un errore altrettanto grave. Tuttavia, a parità di condizioni, non dovrebbero mai essere sacrificate le esigenze dell'utente finale a scapito di una programmazione troppo sofisticata.

## Modificare gli ambienti

Il costo per tenere in attività un computer, espresso come costo per milione di istruzioni al secondo (MIPS), storicamente diminuisce di un tasso che si aggira attorno al venti percento annuo. I costi di lavorazione, d'altro canto, hanno continuato a crescere, non solo a causa delle tendenze generali, ma anche perché i salari dei singoli addetti aumentano in proporzione al tempo trascorso nella stessa azienda e alla professionalità acquisita. Ciò significa che qualsiasi lavoro che possa essere trasferito da operatori umani alle macchine rappresenta un buon investimento.

Come è stato applicato questo incredibile cambiamento alla progettazione di applicazioni? Assolutamente non in maniera uniforme. Il vero progresso si è verificato negli *ambienti*, come accadde inizialmente con il lavoro visionario svolto da Xerox presso il Palo Alto Research Center (PARC), poi su Macintosh e ora MS-Windows, i browser Web e altri sistemi con interfaccia grafica basati su icone. Questi ambienti sono molto più semplici da apprendere e comprendere rispetto a quelli precedenti, basati su comandi scritti, e le persone che li utilizzano riescono a produrre in pochi minuti ciò che prima richiedeva diversi giorni di lavoro. I progressi in alcuni

casi sono stati così rilevanti da far dimenticare quanto determinati compiti risultassero complessi in passato.

Purtroppo, questo concetto di ambiente semplice e agevole non è stato afferrato da molti sviluppatori di applicazioni, che pur lavorando su questi ambienti, non abbandonano le vecchie abitudini ormai inadeguate.

## 2.4 Codici, abbreviazioni e standard di denominazione

Il problema delle abitudini di programmazione ormai radicate diviene evidente soprattutto per i codici, le abbreviazioni e gli standard di denominazione, che vengono completamente ignorati quando si considerano le esigenze degli utenti finali. Quando questi argomenti vengono toccati, in genere vengono esaminate soltanto le esigenze e le convenzioni dei gruppi di sistemi. Potrebbe sembrare una questione sterile e poco interessante, tuttavia può fare la differenza tra un grande successo e un risultato a mala pena accettabile, tra un salto di produttività di prima grandezza e un guadagno marginale, tra utenti interessati ed efficienti e utenti frettolosi che richiedono continuamente l'ausilio dei progettisti.

Ecco cosa è accaduto. Una volta le registrazioni economiche venivano effettuate su libri mastri e giornali. Ciascun evento o transazione veniva trascritto, linea dopo linea, utilizzando la lingua corrente. Durante lo sviluppo delle applicazioni, venivano aggiunti dei codici per sostituire i valori dei dati (come "01" per "Avere", "02" for "Dare" e così via). Gli addetti all'inserimento di questi dati dovrebbero in effetti conoscere o cercare la gran parte di questi codici e digitali nel campo appropriato dello schermo. Si tratta di un esempio estremizzato, tuttavia centinaia di applicazioni operano esattamente con questo approccio e ogni singola parte è ugualmente complessa da imparare o da capire.

Questo problema è stato molto presente nello sviluppo dei grandi sistemi mainframe convenzionali. Quando si introducono i database relazionali in questi gruppi, essi vengono utilizzati semplicemente come parti sostitutive di vecchi metodi di input/output tradizionali come VSAM (Virtual Storage Access Method) e IMS (Information Management System). La potenzialità e le caratteristiche di un database relazionale sono praticamente sprecate, quando vengono utilizzate in questo modo.

### Perché vengono utilizzati i codici e non la lingua corrente?

Perché utilizzare i codici? Sono due le giustificazioni principali generalmente presentate:

- una categoria comprende un numero di articoli così elevato che non possono essere ragionevolmente rappresentati o ricordati utilizzando la lingua corrente;
- per risparmiare spazio di memorizzazione nel computer.

Il secondo punto è anacronistico. La memoria di sistema e la memoria permanente erano talmente costose e le CPU talmente lente (con meno potenza di una calcolatrice tascabile attuale) che i programmati dovevano condensare ogni informazione nel minor spazio possibile. I numeri, a parità di caratteri, occupano la metà dello spazio rispetto alle lettere, e i codici riducono ancora di più le prestazioni richieste alla macchina.

Dato che le macchine erano costose, i progettisti dovettero utilizzare codici per ottenere che ogni cosa funzionasse. Si trattava di una soluzione tecnica a un problema economico. Per gli utenti costretti a imparare ogni genere di codici senza senso, la situazione era pressante. Le

macchine erano troppo lente e troppo costose per soddisfare gli uomini, perciò questi furono addestrati per adattarsi alle macchine. Fu un male necessario.

Questa giustificazione economica svanì anni fa. I computer ora sono sufficientemente economici e veloci per adattarsi al modo di lavorare degli uomini e utilizzare parole comprensibili alle persone. Era ora che accadesse! Eppure, senza porsi il problema consapevolmente, sviluppatori e progettisti continuano volenti o nolenti a utilizzare i codici.

Il primo punto, troppi articoli in ciascuna categoria, è più sostanziale, ma comunque molto meno di quanto sembri a prima vista. Un'idea proposta è quella secondo la quale è necessario minore sforzo (e quindi meno costo) per digitare codici numerici invece dei valori stringa effettivi, come i titoli dei libri. Questa giustificazione non vale nel caso di Oracle. Non soltanto è più costoso addestrare le persone a conoscere il codice corretto per clienti, prodotti, transazioni e altro, senza contare il costo degli errori (che sono molti nei sistemi basati su codici); utilizzare i codici significa anche non utilizzare Oracle in maniera completa. In Oracle è possibile digitare i primi caratteri di un titolo e ottenere che il resto del nome venga inserito automaticamente. La stessa operazione è possibile con nomi di prodotti, transazioni (una "a" può diventare automaticamente "acquisto" e una "v" "vendita") e così via, per tutta l'applicazione. Tutto ciò avviene grazie a sofisticate caratteristiche di ricerca dell'elemento appropriato.

## Il vantaggio del riscontro dell'utente

C'è anche un altro vantaggio: gli errori durante l'inserimento dei dati si riducono quasi a zero, poiché gli utenti hanno un riscontro immediato, in lingua corrente, delle informazioni che vengono inserite. I caratteri digitati non vengono trasposti; i codici non vengono ricordati in maniera errata e nelle applicazioni di carattere finanziario accade raramente che si perda del denaro nei conti a causa di errori di digitazione, con un risparmio davvero significativo.

Anche le applicazioni diventano molto più comprensibili. Le schermate e i report vengono trasformati da arcane serie di numeri e codici in un formato leggibile e comprensibile. Il cambiamento della progettazione di applicazioni da un orientamento che punta sui codici all'utilizzo della lingua naturale ha un effetto profondo e rafforzativo sull'azienda e i suoi impiegati. Sugli utenti che sono stati sommersi da manuali sui codici, la possibilità di lavorare con un'applicazione basata sulla lingua comune produce un effetto altamente rilassante.

## 2.5 Come arginare la confusione

Un'altra versione della giustificazione "troppi articoli per categoria" è quella secondo cui il numero di prodotti, clienti o di tipi di transazione è troppo elevato per riuscire a differenziare ciascun elemento con una denominazione, oppure ci sono troppi articoli all'interno di una categoria che si presentano come identici o molto simili (clienti che si chiamano "Mario Rossi", per esempio). Una categoria può contenere troppe voci tali da rendere difficile l'individuazione delle opzioni o la differenziazione, ma più spesso questo è il risultato di un lavoro incompleto di categorizzazione delle informazioni: troppe cose dissimili sono raccolte in una categoria troppo ampia. Per sviluppare un'applicazione con un forte orientamento all'uso della lingua italiana (o inglese, francese, tedesca e così via), rispetto all'utilizzo di codici, è necessario avere tempo da impiegare con utenti e progettisti separando le informazioni dal lavoro, comprendendone le relazioni e le categorie naturali, e quindi creando con attenzione un database e uno schema di denominazione che rifletta in maniera semplice e accurata queste scoperte.

Le fasi fondamentali di questo processo sono tre:

1. normalizzazione dei dati;
2. scelta delle denominazioni per le tabelle e le colonne;
3. scelta delle denominazioni per i dati.

Ciascuno di questi punti viene descritto seguendo l'ordine indicato. Lo scopo è quello di sviluppare un'applicazione in cui i dati siano organizzati in tabelle e colonne con nomi familiari agli utenti e in cui i dati stessi siano descritti con termini familiari, non con codici.

## Normalizzazione

Le relazioni tra paesi, o tra settori di una società, o tra utenti e progettisti, sono solitamente il prodotto di particolari circostanze storiche, che possono definire le relazioni attuali anche se tali circostanze appartengono al passato remoto. Il risultato può essere quello di relazioni anormali, oppure, secondo una definizione attuale, di disfunzioni. La storia e le circostanze spesso hanno lo stesso effetto sui dati (sulle modalità con cui vengono raccolti, organizzati e riportati). Anche i dati possono diventare anormali e presentare disfunzioni.

La normalizzazione è il processo che serve per riportare le cose alla dimensione corretta, rendendole normali. L'origine del termine è la parola latina *norma*, che indicava una squadra da carpentiere utilizzata per ottenere angoli retti. In geometria, quando una linea forma un angolo retto con un'altra, si definisce "normale" rispetto a questa. In un database relazionale il termine ha anche un preciso significato matematico, che riguarda il concetto di separazione degli elementi dei dati (come nomi, indirizzi o compiti) in gruppi di affinità e che definisce la relazione "normale" tra di essi.

I concetti di base della normalizzazione vengono presentati in questo contesto in modo che gli utenti possano contribuire alla progettazione dell'applicazione che utilizzeranno, o comprendere meglio un'applicazione già impostata. Tuttavia, sarebbe un errore pensare che questo processo sia veramente applicabile soltanto allo sviluppo di un database o ad un'applicazione per computer. I processi di normalizzazione hanno per effetto una comprensione più profonda delle informazioni utilizzate in una procedura e delle modalità di correlazione dei vari elementi che compongono quelle informazioni. Ciò si dimostrerà utile anche in aree diverse da quella dei database e dei computer.

### Il modello logico

Una delle fasi iniziali del processo di analisi è la costruzione di un *modello logico*, che è semplicemente un diagramma normalizzato dei dati utilizzati nella procedura. Conoscere i motivi e le modalità per cui i dati vengono suddivisi e separati è essenziale per comprendere il modello, e quest'ultimo è essenziale per creare un'applicazione che supporti la procedura per un lungo periodo senza dover ricorrere a elementi aggiuntivi.

In genere si parla di normalizzazione in termini di *forma*: la prima, la seconda e la terza forma normale sono le più comuni, laddove la terza rappresenta lo status più altamente normalizzato. Esistono anche delle definizioni per il quarto e il quinto livello di normalizzazione, tuttavia non rientrano nell'ambito della nostra discussione.

Si consideri una biblioteca; per ciascun libro è possibile memorizzare informazioni come titolo, autore, editore e diverse categorie di termini descrittivi a esso relativi. Si supponga che questi dati relativi ai libri diventino la struttura di una tabella in Oracle. La tabella potrebbe essere chiamata BIBLIOTECA, mentre le colonne potrebbero essere Titolo, Editore, Autore1, Autore2, Autore3, Categoria1, Categoria2 e Categoria3. Gli utenti di questa tabella hanno già un problema: nella tabella BIBLIOTECA è possibile elencare solo tre autori o categorie per un singolo libro.

Cosa accade quando l'elenco delle categorie accettabili cambia? Qualcuno sarà costretto a scorrere tutte le categorie della tabella BIBLIOTECA e correggere i valori precedenti. E se uno degli autori cambia nome? Ancora una volta è necessario modificare tutti i record correlati. E nel caso in cui ci sia un quarto autore che contribuisce alla stesura di un libro?

I problemi appena elencati non sono questioni legate all'uso del computer o tecniche, anche se vengono alla luce durante la progettazione di un database. Si tratta soprattutto di questioni di base sulle modalità di organizzare in modo attento e logico le informazioni relative a un dato compito, mediante procedimenti di normalizzazione. Occorre riorganizzare passo dopo passo gli elementi dei dati in gruppi di affinità, eliminando le relazioni non funzionali e stabilendo relazioni normali.

### **Normalizzazione dei dati**

Il primo passo verso la riorganizzazione consiste nell'impostare i dati nella prima forma normale. A tal fine si procede spostando i dati in tabelle separate, dove vengono raggruppati per tipo, e si assegna a ciascuna tabella una *chiave primaria* (un'etichetta o un identificativo specifico). In questo modo viene eliminata la ripetizione di gruppi di dati, come gli autori presenti nella biblioteca.

Invece di mantenere tre autori per libro, i dati di ciascuno sono collocati in una tabella separata, con una riga per nome e descrizione. In questo modo viene eliminata la necessità di un numero variabile di autori nella tabella BIBLIOTECA e si procede in modo più funzionale rispetto a limitare la tabella BIBLIOTECA a tre soli autori.

Successivamente occorre definire la chiave primaria di ciascuna tabella: quale elemento può identificare una tabella in maniera univoca e consentire di estrarlarne una riga di informazioni? Per semplicità, si presume che i titoli e i nomi degli autori siano unici, quindi NomeAutore è la chiave primaria per la tabella BIBLIOTECA.

A questo punto la tabella BIBLIOTECA è suddivisa in due tabelle: AUTORE, con le colonne NomeAutore (la chiave primaria) e Commenti, e BIBLIOTECA, con la chiave primaria Titolo e con le colonne Editore, Categoria1, Categoria2, Categoria3, Classificazione e DescrizioneClassificazione. Una terza tabella, AUTORE\_BIBLIOTECA, fornisce le associazioni: è possibile elencare più autori per uno stesso libro e un autore può scrivere più libri: in questo caso si parla di relazione molti a molti. Nella Figura 2.1 è possibile vedere queste relazioni e le chiavi primarie.

Nel successivo passaggio del processo di normalizzazione, la seconda forma normale, è richiesta la selezione dei dati che sono dipendenti soltanto da una parte della chiave. Se ci sono attributi che non dipendono totalmente da una chiave, devono essere spostati in una nuova tabella. In questo caso DescrizioneClassificazione non dipende effettivamente da Titolo, in quanto si basa sul valore della colonna Classificazione; per questo motivo dev'essere spostata in una tabella separata.

Nell'ultimo passaggio, la terza forma normale, occorre sbarazzarsi di tutti gli elementi delle tabelle che non sono dipendenti unicamente dalla chiave primaria. In questo esempio le categorie sono intercorrelate; un titolo non verrà elencato sia come "Narrativa" sia come "Saggistica", e la categoria "Adulti" avrà sottocategorie diverse da quelle presenti sotto la categoria "Ragazzi". Le informazioni di categoria vengono pertanto spostate in una tabella separata. Nella Figura 2.2 sono illustrate le tabelle nella terza forma normale.

Ogniqualvolta i dati sono impostati nella terza forma normale, sono automaticamente impostati anche nella seconda e nella prima. Pertanto non è necessario passare da una forma all'altra per attuare l'intero processo. È sufficiente impostare i dati in modo che le colonne di ciascuna tabella, invece che essere dipendenti dalla chiave primaria, siano dipendenti soltanto dalla *chiave primaria completa*. La terza forma normale può essere quindi definita come "la chiave, la chiave completa e nient'altro che la chiave".

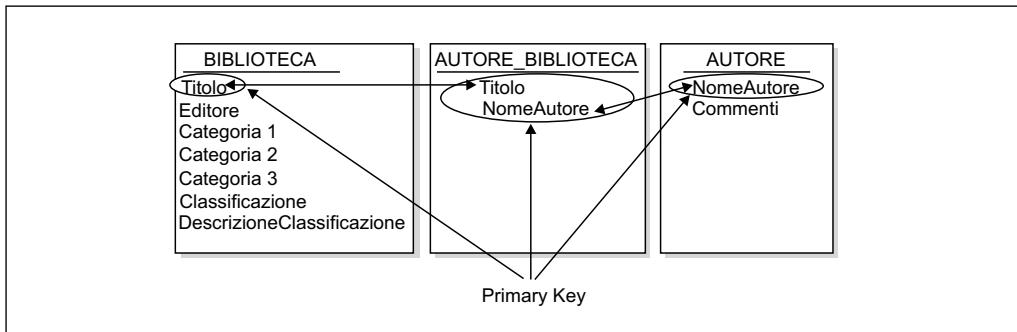


Figura 2.1 Le tabelle BIBLIOTECA, AUTORE e AUTORE\_BIBLIOTECA.

### Navigare tra i dati

Il database della biblioteca ora è nella terza forma normale. Nella Figura 2.3 viene presentato un campione di quello che potrebbero contenere queste tabelle. La correlazione tra queste tabelle risulta immediatamente evidente. Si può navigare da una tabella all'altra per estrapolare informazioni su un determinato autore, sulla base delle chiavi impostate per ciascuna tabella. In ogni tabella, con la chiave primaria è possibile identificare in maniera univoca una singola riga. Per esempio, se si seleziona Jay Gould, si può facilmente scoprirne il record nella tabella AUTORE, perché NomeAutore è la chiave primaria di questa tabella.

Cercando Harper Lee nella colonna NomeAutore della tabella AUTORE\_BIBLIOTECA si vedrà che ha pubblicato un romanzo dal titolo “To Kill A Mockingbird”. A questo punto è possibile controllare l'editore, la categoria e la classificazione di questo libro nella tabella BIBLIOTECA e consultare la tabella CLASSIFICAZIONE per trovare una descrizione della classificazione.

Quando si cerca “To Kill A Mockingbird” nella tabella BIBLIOTECA, si effettua una ricerca in base alla chiave primaria di quest'ultima. Per trovare l'autore di questo libro, è sufficiente invertire il percorso di ricerca precedente, cercando nella tabella AUTORE\_BIBLIOTECA i

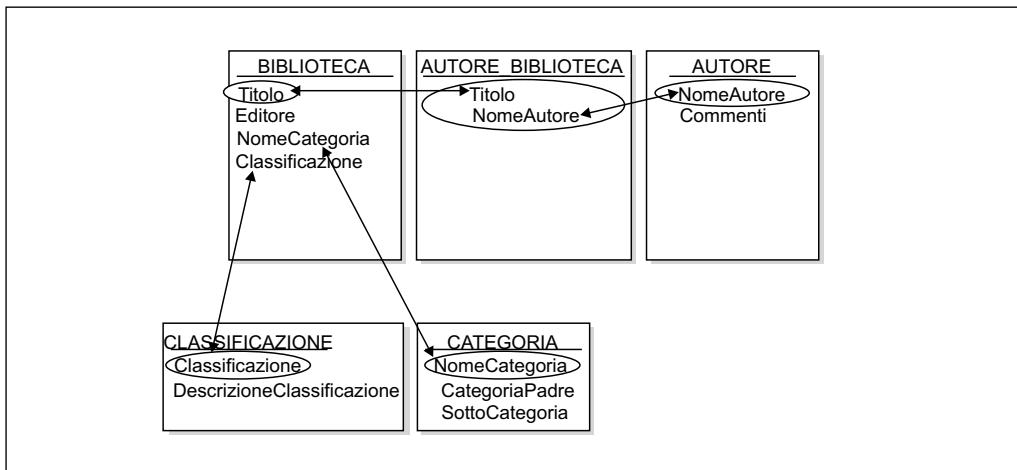


Figura 2.2 La tabella BIBLIOTECA e le tabelle a essa correlate.

record che hanno quel valore nella colonna Titolo: questa colonna rappresenta una chiave esterna per la tabella AUTORE\_BIBLIOTECA. Quando la chiave primaria per BIBLIOTECA appare in un'altra tabella, come avviene nella tabella AUTORE\_BIBLIOTECA, essa prende il nome di *chiave esterna* per quella tabella.

Osservando queste tabelle si possono anche trarre conclusioni sul mondo reale: ci sono classificazioni e categorie non ancora utilizzate dai libri presenti nella biblioteca. Dato che i dati sono organizzati in modo logico, è possibile mantenere un record delle categorie, delle classificazioni e degli autori potenziali anche se nessuno dei libri attualmente presenti utilizza tali valori.

Si tratta di una forma di organizzazione dell'informazione di tipo logico e attento, anche se le "tabelle" sono scritte in un libro mastro o su schede cartacee rudimentali. Naturalmente occorre ancora qualche aggiustamento per trasformare il tutto in un vero database. Per esempio, NomeAutore probabilmente dovrebbe essere suddiviso in Nome e Cognome, e si potrebbe voler trovare un modo per indicare l'autore principale oppure il fatto che ci sia un curatore al posto di un autore.

Tutto questo processo viene definito *normalizzazione* e non è più complesso di quanto sia stato illustrato finora. Per ottenere un buon risultato occorre analizzare anche altre questioni, tuttavia gli elementi fondamentali dell'analisi delle relazioni "normali" tra i vari elementi dei dati sono effettivamente semplici come quelli appena presentati. La vera discriminante è costituita dall'utilizzo o meno di un database relazionale o di un computer.

In ogni caso occorre una precisazione importante. La normalizzazione fa parte del processo di analisi, non è la progettazione, che comprende molte altre considerazioni, ed è un errore grave credere che le tabelle normalizzate del modello logico rappresentino il "progetto" effettivo per un database. Questa confusione di fondo tra analisi e progetto contribuisce a creare le storie riportate dalla stampa sul fallimento di grandi applicazioni relazionali. Questi problemi vengono trattati in maniera specifica per gli sviluppatori più avanti nel capitolo.

## Nomi in lingua corrente per tabelle e colonne

Una volta comprese e correttamente impostate le relazioni tra i diversi elementi che costituiscono i dati di un'applicazione, occorre dedicare la propria attenzione alla scelta dei nomi per le tabelle e le colonne in cui devono essere inseriti i dati stessi. Si tratta di un tema al quale viene data troppo poca importanza, anche da parte degli esperti. Le denominazioni di tabella e di colonna vengono spesso scelte senza consultare gli utenti finali e senza una revisione rigorosa. Entrambe queste mancanze possono avere serie conseguenze quando viene il momento di utilizzare effettivamente l'applicazione.

Si considerino per esempio le tabelle della Figura 2.3. I nomi della tabella e delle colonne sono pressoché tutti di comprensione immediata. Un utente finale, anche se nuovo a concetti relazionali e SQL, avrebbe poche difficoltà per comprendere o anche per riprodurre una query come questa:

```
select Titolo, Editore
  from BIBLIOTECA
 order by Editore;
```

Gli utenti sono in grado di comprendere il codice perché le parole utilizzate risultano familiari, non si tratta di termini oscuri o mal definiti. Quando occorre definire tabelle con molte più colonne, denominare queste ultime può essere più difficile, tuttavia alcuni principi cardine risultano di grandissima utilità. Si considerino alcune delle difficoltà comunemente causate dalla mancanza di convenzioni sui nomi. Che cosa accadrebbe se fossero stati scelti i nomi seguenti?

AUTORE	NomeAutore	Commenti		
DIETRICH BONHOEFFER		TEOLOGO TEDESCO, UCCISO IN CAMPO DI CONCENTRAMENTO		
ROBERT BRETALL		STUDIOSO DI KIERKEGAARD		
ALEXANDRA DAY		AUTRICE DI LIBRI ILLUSTRATI PER BAMBINI		
STEPHEN JAY GOULD		GIORNALISTA SCIENTIFICO, PROFESSORE AD HARVARD		
SOREN KIERKEGAARD		FILOSOFO E TEOLOGO DANESI		
HARPER LEE		ROMANZIERE AMERICANO, HA PUBBLICATO UN SOLO ROMANZO		
LUCY MAUD MONTGOMERY		SCRITTRICE CANADESE		
JOHN ALLEN PAULOS		PROFESSORE DI MATEMATICA		
J. RODALE		ESPERTO DI GIARDINAGGIO		
CLASSIFICAZIONE	Classificazione	DescrizioneClassificazione		
1	EVASIONE			
2	INFORMAZIONI DI BASE			
3	CONSIGLIATO			
4	VIVAMENTE CONSIGLIATO			
5	LETTURA INDISPENSABILE			
CATEGORIA	NomeCategoria	CategoriaPadre	SottoCategoria	
CONSADULTI	ADULTI		CONSULTAZIONE	
NARRADULTI	ADULTI		NARRATIVA	
SAGGIADULTI	ADULTI		SAGGI	
ILLUSRAGAZZI	RAGAZZI		ILLUSTRATI	
NARRRAGAZZI	RAGAZZI		NARRATIVA	
SAGGIRAGAZZI	RAGAZZI		SAGGI	
AUTORE_BIBLIOTECA	Titolo	NomeAutore		
TO KILL A MOCKINGBIRD		HARPER LEE		
WONDERFUL LIFE		STEPHEN JAY GOULD		
INNUMERACY		JOHN ALLEN PAULOS		
KIERKEGAARD ANTHOLOGY		ROBERT BRETALL		
KIERKEGAARD ANTHOLOGY		SOREN KIERKEGAARD		
ANNE OF GREEN GABLES		LUCY MAUD MONTGOMERY		
GOOD DOG, CARL		ALEXANDRA DAY		
LETTERS AND PAPERS FROM PRISON		DIETRICH BONHOEFFER		
BIBLIOTECA	Titolo	Editore	NomeCategoria	Classificazione
TO KILL A MOCKINGBIRD		HARPERCOLLINS	NARRADULTI	5
WONDERFUL LIFE		W.W.NORTON & CO.	SAGGIADULTI	5
INNUMERACY		VINTAGE BOOKS	SAGGIADULTI	4
KIERKEGAARD ANTHOLOGY		PRINCETON UNIV PR	CONSADULTI	3
ANNE OF GREEN GABLES		GRAMERCY	NARRRAGAZZI	3
GOOD DOG, CARL		LITTLE SIMON	ILLUSRAGAZZI	1
LETTERS AND PAPERS FROM PRISON		SCRIBNER	SAGGIADULTI	4

Figura 2.3 Esempio di dati tratti da BIBLIOTECA e dalle tabelle correlate.

BIBLIOTECA	A_L	AUT	CATEGORIE
titolo	titolo	autn	cat
ed	autn	comm	cat_p
cat			cat_s
clas			

Le tecniche di denominazione utilizzate in questa tabella, per quanto appaiano bizzarre, sono purtroppo molto comuni. Rappresentano tabelle e colonne denominate seguendo le convenzioni utilizzate da parecchi produttori e progettisti noti.

Ecco alcuni dei problemi più ovvi che si presentano in questo elenco di nomi.

- *Abbreviazioni utilizzate arbitrariamente.* Ricordare il nome di una tabella o di una colonna risulta pressoché impossibile. I nomi potrebbero anche essere dei codici, poiché in entrambi i casi occorre decifrarli.
- *Abbreviazioni incongruenti.*
- *Lo scopo o il significato di una colonna o tabella non risulta evidente dalla denominazione.* Oltre al fatto che le abbreviazioni utilizzate rendono difficile ricordare i nomi, anche la natura dei dati contenuti nella colonna o nella tabella risulta oscura. Che cosa significa cat\_p? E comm?
- *Trattini di sottolineatura incongruenti.* Talvolta sono utilizzati per distinguere le parole in un nome, altre volte no. Come è possibile che qualcuno ricordi quale nome è o non è separato da un trattino di sottolineatura?
- *Uso incongruente del plurale.* Anche se non direttamente visibile in questo esempio, si tratta di un problema che viene alla luce molto di frequente.
- *Le regole apparentemente impiegate hanno dei limiti evidenti.* Se, per esempio, la prima lettera della denominazione di tabella viene utilizzata come nome di colonna, che cosa accade quando è necessario utilizzare una tabella che inizia con la stessa lettera? Anche in questo caso viene assegnato un nome di colonna uguale?

Queste sono solo alcune delle difficoltà più ovvie. Gli utenti costretti a questa denominazione scarna di tabelle e colonne non saranno in grado di digitare query semplici in lingua corrente. Le query non avranno quella qualità intuitiva e familiare che ha la tabella BIBLIOTECA e in questo modo verranno notevolmente minate l'accettazione e l'utilità dell'applicazione.

In passato era richiesto che i nomi fossero lunghi al massimo sei o otto caratteri. Come risultato, i nomi erano inevitabilmente un miscuglio confuso di lettere, numeri e abbreviazioni criptiche. Come molte altre limitazioni della tecnologia precedente, questo modo di operare non è più necessario. Con Oracle è consentito utilizzare nomi di tabella e di colonna che contengono fino a 30 caratteri. In questo modo gli utenti hanno a disposizione moltissimo spazio per creare nomi completi, non ambigui e descrittivi.

Per le difficoltà elencate è possibile trovare facilmente delle soluzioni: evitare abbreviazioni e plurali, eliminare trattini di sottolineatura o utilizzarli in modo coerente. Queste semplici norme pratiche consentono di risolvere la confusione oggi prevalente nelle modalità di denominazione. Le convenzioni di denominazione devono essere contemporaneamente semplici, facili da comprendere e da ricordare. In un certo senso, occorre applicare una normalizzazione dei nomi. In maniera molto simile a quella in cui i dati sono analizzati logicamente, separati in base allo scopo e perciò normalizzati, anche gli standard di denominazione dovrebbero avere lo stesso tipo di attenzione logica, altrimenti il compito di creare un'applicazione viene esplicato in maniera impropria.

## Parole in lingua corrente per i dati

Dopo aver sollevato l'importante questione delle convenzioni di denominazione, il passo successivo consiste nell'analizzare direttamente i dati stessi. Dopo tutto, quando i dati delle tabelle vengono stampati su un report, il loro grado di chiarezza determina la comprensibilità del rapporto stesso. Nell'esempio della tabella BIBLIOTECA, Classificazione è un valore di codice, mentre una Categoria è una concatenazione di più valori. È un miglioramento? Se si chiede a una terza persona informazioni su un libro, è accettabile la risposta che ha una classificazione pari a 4 in NarrAdulti? Perché dovrebbe essere permesso a una macchina di essere meno chiara?

Inoltre, avere informazioni in lingua corrente rende più facile impostare e capire le query. La query dovrebbe essere il più possibile simile a una domanda in lingua corrente.

```
select Titolo, NomeAutore
  from AUTORE_BIBLIOTECA;
```

Titolo	NomeAutore
TO KILL A MOCKINGBIRD	HARPER LEE
WONDERFUL LIFE	STEPHEN JAY GOULD
INNUMERACY	JOHN ALLEN PAULOS
KIERKEGAARD ANTHOLOGY	ROBERT BRETALL
KIERKEGAARD ANTHOLOGY	SOREN KIERKEGAARD
ANNE OF GREEN GABLES	LUCY MAUD MONTGOMERY
GOOD DOG, CARL	ALEXANDRA DAY
LETTERS AND PAPERS FROM PRISON	DIETRICH BONHOEFFER

## 2.6 Maiuscole e minuscole nei nomi e nei dati

Con Oracle è un po' più semplice ricordare i nomi di tabella e di colonna, poiché non è rilevante la distinzione tra caratteri maiuscoli o minuscoli. Tali denominazioni sono memorizzate nel dizionario interno dei dati in caratteri maiuscoli. Quando si digita una query, i nomi di tabella e di colonna vengono istantaneamente convertiti in maiuscolo e quindi verificati nel dizionario. Per alcuni altri sistemi relazionali la distinzione tra maiuscole e minuscole è rilevante. Se gli utenti digitano un nome di colonna come "Capacita", ma per il database quel nome è "capacita" o "CAPACITA" (a seconda di come è stato impostato al momento della creazione della tabella) la query non risulterà comprensibile.

**NOTA** È possibile costringere Oracle a creare tabelle e colonne con nomi che contengono lettere sia maiuscole sia minuscole, tuttavia questa pratica rende più complessa la formulazione delle query e il lavoro con i dati. Si consiglia pertanto di utilizzare il comportamento predefinito, che prevede l'uso del maiuscolo.

La capacità di dar vita a tabelle che distinguono tra maiuscole e minuscole viene indicata come un vantaggio poiché, per esempio, permette ai programmatore di creare molte tabelle con nomi simili. Si può creare una tabella lavoratore, una Lavoratore, una IAVORAtore e così via, all'infinito. Sono tutte tabelle diverse. Come si può supporre però che qualcuno, compreso il programmatore, sia in grado ricordare le differenze? In realtà è un difetto, non un vantaggio, e i progettisti di Oracle sono stati abili nel non cadere in questa trappola.

Lo stesso discorso si può fare per i dati memorizzati in un database. Esistono modalità di ricerca delle informazioni in un database senza distinzioni tra caratteri maiuscoli o minuscoli,

ma con questi metodi viene imposto un fardello non necessario. Tranne poche eccezioni, come i testi di carattere legale o paragrafi di lettere modulari, è molto più semplice memorizzare i dati nel database in caratteri maiuscoli. In questo modo le query sono più semplici e i report hanno un aspetto più coerente. Se e quando è necessario che alcuni dati vengano digitati in minuscolo, o utilizzando sia caratteri maiuscoli sia minuscoli (come nel caso di nome e indirizzo su una lettera), allora si può ricorrere alle funzioni di Oracle che effettuano la conversione dei caratteri. È generalmente meno complicato e crea meno confusione memorizzare e visualizzare i dati in caratteri maiuscoli.

Se si ritorna indietro in questo capitolo, si osserverà che questa prassi non è stata seguita. Da questo punto in poi, ora che l'argomento è stato introdotto e inquadrato nel contesto appropriato, i dati del database saranno riportati in caratteri maiuscoli, con l'eccezione di una o due tabelle e alcuni casi isolati.

## 2.7 Normalizzazione dei nomi

Sul mercato esistono molti prodotti in cui si utilizza il “linguaggio naturale” e il cui scopo è quello di consentire la creazione di query utilizzando termini comuni invece di vecchi conglomerati. Questi prodotti funzionano creando una mappa logica con i termini di uso comune e i nomi di colonna, di tabella e i codici in termini non di uso comune e difficili da ricordare. La creazione di tali mappe richiede molta attenzione, ma una volta completata, consente all'utente di interagire facilmente con l'applicazione. Perché non porre la stessa attenzione all'inizio della progettazione? Perché creare la necessità di un'altra stratificazione, di un altro prodotto e più lavoro, quando gran parte della confusione avrebbe potuto essere evitata semplicemente con un buon lavoro di denominazione iniziale?

Per motivi legati al miglioramento delle prestazioni può accadere che alcuni dei dati di un'applicazione siano ancora memorizzati in maniera codificata all'interno del database del computer. Questi codici *non* dovrebbero essere alla portata degli utenti, sia nella fase di inserimento dei dati, sia nella fase di estrapolazione, e con Oracle è molto facile mantenerli nascosti.

Nel momento in cui i dati richiedono l'utilizzo di codici, gli errori nell'impostazione delle chiavi aumentano. Quando un report contiene codici invece di espressioni in lingua comune, iniziano gli errori di interpretazione. E quando gli utenti hanno bisogno di creare nuovi report, o report ad hoc, la loro possibilità di farlo in maniera veloce e precisa risulta gravemente compromessa sia dalla necessità di utilizzare codici, sia dal fatto di non essere in grado di ricordare nomi di colonna e di tabella stravaganti.

Con Oracle gli utenti hanno la possibilità di visualizzare i dati e operare utilizzando la lingua comune in tutte le fasi dell'applicazione. Ignorare questa opportunità significa sprecare le potenzialità di Oracle e senza dubbio produrre un'applicazione meno comprensibile e meno produttiva. I progettisti dovrebbero cogliere questa opportunità e gli utenti dovrebbero prenderla. Entrambe le categorie ne trarranno enormi benefici.

## 2.8 L'importanza del fattore umano

A questo punto, chi è nuovo del mondo Oracle potrebbe voler iniziare a lavorare direttamente con Oracle e il linguaggio SQL. Questo argomento verrà trattato nel prossimo capitolo, mentre nella parte restante di questo ci si concentrerà su diverse considerazioni relative alle prestazioni, alla denominazione e alla progettazione. Si potrà sempre fare riferimento a questo paragrafo in un secondo momento, quando ci si sentirà pronti per progettare e implementare un'applicazione.

Questo paragrafo del capitolo è dedicato a stabilire quale metodo di approccio utilizzare per un progetto di sviluppo che tenga in considerazione le operazioni commerciali effettive che l'utente finale dev'essere in grado di eseguire. Tutto ciò differisce dal più comune orientamento ai dati di molti sviluppatori e di numerose tecnologie di sviluppo. La normalizzazione dei dati e le tecnologie CASE (Computer Aided Software Engineering) sono diventate a tal punto il centro d'attrazione nello sviluppo di applicazioni relazionali, che il concentrarsi sui dati e sui problemi di integrità referenziale, sulle chiavi, sulla normalizzazione e sui diagrammi delle tabelle è diventato quasi un'ossessione. Tutti questi elementi vengono spesso confusi con la progettazione e, a volte, viene spesso accolto con sorpresa il fatto che ciò che si crede essere progettazione si tratta, in realtà, di analisi.

*La normalizzazione è analisi, non progettazione.* Essa rappresenta solo una parte dell'analisi necessaria per capire un'operazione commerciale e per costruire un'applicazione utile. Lo scopo dello sviluppo di un'applicazione, in fondo, è quello di aiutare ad avere maggior successo negli affari; l'obiettivo viene raggiunto migliorando la velocità e l'efficienza nell'adempiere le operazioni commerciali e anche rendendo l'ambiente nel quale si lavora il più possibile significativo e di aiuto. Date alle persone il controllo sulle proprie informazioni e un accesso intuitivo e diretto a esse, e quelle persone risponderanno con gratitudine e produttività. Assegnate il controllo a un gruppo remoto, rendete vaghe le informazioni per mezzo di codici e interfacce ostili agli utenti, e le stesse persone saranno infelici e improduttive.

I metodi descritti in questo paragrafo non sono intesi come una trattazione rigorosa del processo, e gli strumenti di trattamento delle strutture e dei flussi di dati che si utilizzano, e con cui si ha familiarità, sono probabilmente sufficienti per i propri scopi. Lo scopo che ci si prefigge è quello di descrivere un approccio efficace per la creazione di applicazioni rapide, adeguate e adattabili.

## Comprensione dei compiti dell'applicazione

Uno dei passi spesso trascurati nella realizzazione del software è quello di una profonda comprensione di quale sia il lavoro dell'utente finale, ovvero dei compiti che l'automazione computerizzata è chiamata ad agevolare. Occasionalmente ciò avviene in quanto l'applicazione stessa è piuttosto specializzata. Più spesso ciò accade perché l'approccio alla progettazione tende a essere orientato ai dati da trattare. Frequentemente, durante l'analisi vengono posti come principali quesiti quelli di seguito elencati.

- Quali dati devono essere catturati?
- Come devono essere elaborati i dati?
- Come devono essere restituiti i dati?

Tali domande si diramano in una serie di domande secondarie, che chiamano in causa problemi quali i moduli di inserimento dei dati, la disposizione dello schermo, i calcoli, l'invio di messaggi, le correzioni, le sessioni di revisione, la ritenzione dei dati, i volumi di memorizzazione, i cicli di elaborazione, la formattazione dei report, la distribuzione e la manutenzione. Tutte le aree elencate sono di importanza vitale. Una difficoltà隐式的 è però il fatto che tali aree si concentrano esclusivamente sui problemi relativi ai dati.

Le persone utilizzano i dati, ma per svolgere dei compiti. Qualcuno può obiettare che, benché ciò sia vero per i professionisti, gli impiegati che inseriscono i dati si limitano a trasferirli da un modulo d'inserimento a una tastiera. Il loro compito è pertanto molto orientato ai dati. Questo è un ritratto attendibile del tipo odierno di lavoro. Ma ciò è una conseguenza del lavoro reale che dev'essere eseguito, o è una conseguenza della modalità di progettazione delle applicazioni per computer? L'uso di esseri umani come periferiche di inserimento dei dati, in particolare per

dati voluminosi, di formato regolare (come avviene nei moduli) e con un ambito di variazione limitato, rappresenta un costoso e antiquato, per non dire disumano, metodo di raccolta dei dati. Come è successo per l'utilizzo di codici per adattarsi ai limiti della macchina, questa idea ha fatto il suo tempo.

Quanto esposto può suonare come semplice filosofia, tuttavia ha una ricaduta pratica sul modo in cui viene intrapresa la progettazione delle applicazioni. Le persone utilizzano i dati, ma per svolgere dei compiti, e non possono portare a termine tali compiti uno per volta. Le persone svolgono numerosi compiti che sono fasi secondarie gli uni degli altri, o che si intersecano tra loro e devono svolgerli contemporaneamente, in parallelo.

Quando i progettisti acconsentono a fare in modo che sia questa idea a guidare l'analisi e la creazione di un'applicazione, invece di concentrarsi sull'orientamento ai dati che è storicamente stato dominante, l'autentica natura degli sforzi compiuti cambia in modo significativo. Gli ambienti a finestre hanno avuto il ben noto successo in quanto consentono all'utente di passare rapidamente attraverso piccoli programmi, mantenendoli tutti attivi contemporaneamente senza che sia necessario uscire da un programma per poterne avviare un altro. L'ambiente a finestre si avvicina a una rappresentazione di come le persone in effetti pensano e lavorano più di quanto non lo facesse il vecchio buon approccio di tipo "una cosa per volta". La lezione che se ne ricava non deve andare perduta. È su tale lezione, infatti, che si deve costruire.

La comprensione dei compiti dell'applicazione significa andare ben oltre l'identificazione degli elementi relativi ai dati, la loro normalizzazione e la creazione di finestre, programmi di elaborazione e report. La comprensione delle funzioni dell'applicazione significa comprendere che cosa gli utenti fanno e quali siano i loro compiti, in modo da progettare un'applicazione che risponda a tali esigenze, e non alla semplice esigenza di raccogliere i dati. In effetti, quando l'orientamento è verso i dati, il progetto risultante distorce inevitabilmente i compiti dell'utente anziché agevolarli.

Nel progettare applicazioni più reattive ai compiti che non ai dati, il trucco migliore consiste nel comprendere semplicemente che ciò è necessario. In questo modo, l'approccio all'analisi di un'applicazione economica è vista da una prospettiva nuova.

Il primo passo nell'analisi procedurale consiste nel comprendere i compiti da svolgere. Qual è il compito per cui i membri di questo gruppo hanno effettivamente bisogno di utilizzare il computer? Qual è il vero servizio o bene prodotto? Queste possono sembrare domande fondamentali e persino semplicistiche, ma un numero sorprendentemente alto di uomini d'affari è decisamente poco chiaro nelle risposte. Un gran numero di settori d'impresa, dalla sanità al sistema bancario, dalle spedizioni alla produzione, è convinta di operare nel settore dell'elaborazione dei dati. Dopo tutto, in tali settori si inseriscono dati nei computer, li si elabora e se ne traggono delle relazioni. Questo fraintendimento non è che un altro sintomo dell'orientamento ai dati della progettazione dei nostri sistemi, orientamento che ha condotto decine di aziende a tentare di commercializzare ciò che esse ritenevano essere il "vero" prodotto, l'elaborazione dei dati, con risultati disastrosi per la maggior parte di esse.

Da ciò l'importanza di apprendere ciò che riguarda un'applicazione commerciale. Occorre mantenere una buona apertura mentale, e spesso è necessario sfatare le opinioni preferite di qualcuno sulla natura degli affari trattati, per scoprire la vera essenza di tale natura. Si tratta di un processo utile, seppure a volte difficoltoso.

E proprio come è essenziale che gli uomini d'affari divengano utenti esperti del linguaggio SQL e comprendano i fondamenti del modello relazionale, è altrettanto importante che i progettisti di applicazioni comprendano realmente il servizio o prodotto fornito e i compiti necessari per realizzarlo. Un gruppo di progettazione che comprenda utenti finali ai quali sono stati impartiti i fondamenti del linguaggio SQL e dell'approccio relazionale, per esempio grazie alla lettura di questo volume, e progettisti che siano sensibili alle esigenze degli utenti finali e comprendano il

valore di un ambiente applicativo orientato ai compiti da svolgere e sviluppato in un linguaggio leggibile, consente di realizzare sistemi di qualità straordinariamente alta. I membri di un gruppo di sviluppo di questo genere supporteranno e apporteranno miglioramenti agli sforzi degli altri componenti.

Un tipo di approccio a questo processo prevede lo sviluppo di due documenti convergenti; un documento dei compiti e un documento dei dati. È nella fase di preparazione della documentazione dei compiti che entra in gioco la profonda comprensione dell'applicazione. La documentazione dei dati agevola l'implementazione della visione e assicura che si tengano presenti tutti i dettagli e le regole, mentre il documento dei compiti definisce la visione di quale sia la natura degli affari trattati.

## Profilo dei compiti

Il documento dei compiti rappresenta lo sforzo congiunto degli utenti e di chi ha progettato l'applicazione. In tale documento, vengono elencati in ordine tutti i compiti associati alle operazioni commerciali trattate. Il documento inizia con una descrizione schematica del tipo di affari trattati. Tale descrizione dev'essere costituita da una semplice frase dichiarativa composta da un numero di parole che varia da tre a dieci, in prima persona, senza virgolette e con il minimo possibile di aggettivi. Per esempio:

*Noi vendiamo assicurazioni.*

La frase non dev'essere assolutamente simile alla seguente.

*La Amalgamated Diversified è un fornitore internazionale leader di risorse finanziarie, addestramento, elaborazione di informazioni, raccolta e distribuzione di transazioni, comunicazioni, supporto ai clienti e gestione industriale nel campo del rischio condiviso, impegnata in ambito sanitario, nella conservazione delle proprietà e nella responsabilità civile per le auto.*

C'è un'enorme tentazione di stipare in questa prima frase ogni più piccolo dettaglio sugli affari trattati e sui sogni relativi. Questo non è il comportamento più opportuno. Lo sforzo di tagliare gli eccessi descrittivi sino a ottenere una frase semplice mette meravigliosamente ordine nella mente. Se non si è in grado di descrivere il lavoro in dieci parole, non lo si è ancora compreso.

Il progetto dell'applicazione non è chiamato a creare da solo questa frase: il compito verrà svolto in collaborazione con l'utente e darà inizio alla fase di documentazione dei compiti da svolgere. La frase sintetica descrittiva dà l'opportunità di iniziare a riflettere seriamente su quali siano gli affari trattati e su come vengano svolti. Questa fase è di grande valore per gli affari stessi, a prescindere del fatto che ne venga realizzata un'applicazione. In questo modo si affrontano numerosi compiti principali e secondari, procedure e regole che alla prova si dimostreranno senza significato o di importanza molto marginale. Normalmente questi sono retaggi sia di precedenti problemi, risolti da tempo, o di informazioni, o richieste di relazioni, da parte di dirigenti da tempo non più presenti.

Qualche burlone ha suggerito che il metodo per risolvere il problema della creazione di un numero troppo alto di relazioni sia quello di smettere semplicemente di produrle e stare a vedere se qualcuno se ne accorge. Questa è una battuta, ma il nocciolo di verità in essa contenuto dovrà fare parte del procedimento di documentazione dei compiti. Esso infatti si è dimostrato piuttosto

utile negli sforzi per rimediare al problema del passaggio all'anno 2000, in quanto per molti programmi e report non è stato necessario apportare correzioni perché non erano più utilizzati.

L'approccio della cerchia di persone che si sforzano di documentare i compiti da svolgere consente di porre domande decisamente scettiche e individuare (per valutarne l'utilità) ciò che potrebbe essere un semplice retaggio. Il progettista deve comunque essere consci del fatto che, appunto in quanto tale, non può comprendere gli affari trattati nel modo approfondito in cui li conosce l'utente. C'è una linea di demarcazione ben precisa tra afferrare l'opportunità, durante lo sviluppo di un'applicazione, per razionalizzare i compiti svolti e spiegarne i motivi, e rischiare di offendere gli utenti presumendo di comprendere la "vera" natura dei loro affari meglio di quanto essi stessi sappiano.

È necessario chiedere all'utente di descrivere ciascun compito in dettaglio e di spiegare la ragione di ogni singolo passo. Se tale ragione è debole, quale "abbiamo sempre fatto così" o "qualcuno utilizzerà pure questa cosa", deve scattare il semaforo rosso. È consigliabile dire chiaramente che non si capisce e chiedere una nuova spiegazione. Se la risposta resta insoddisfacente, si annoti il compito e la domanda posta su un elenco separato, per una successiva risoluzione. Ad alcune delle domande verrà risposto semplicemente da qualcuno che conosce meglio la materia, altre richiederanno colloqui con i dirigenti più esperti, alla fine numerosi compiti verranno eliminati in quanto non più necessari. Una prova della validità di un procedimento di analisi è il miglioramento delle procedure esistenti, a prescindere dall'implementazione di una nuova applicazione informatica, e normalmente molto prima di essa.

### **Formato generale della documentazione dei compiti**

Di seguito viene descritto il formato generale del documento dei compiti.

- Frase riassuntiva che descriva gli affari trattati (da tre a dieci parole).
- Frasi riassuntive che descrivano e numerino i compiti principali relativi agli affari trattati (frasi e parole brevi).
- Descrizione dettagliata di compiti di livelli aggiuntivi, a seconda della necessità, all'interno di ciascuno dei compiti principali.

È possibile far seguire la frase riassuntiva di ciascun livello da una breve annotazione descrittiva, ma ciò non deve essere utilizzato come scusa per evitare lo sforzo di rendere la frase riassuntiva stessa chiara e incisiva. I compiti principali vengono normalmente numerati con 1.0, 2.0, 3.0 e via dicendo, e a questi ci si riferisce a volte come a compiti di livello zero. I livelli sottostanti vengono numerati utilizzando punti aggiuntivi, come in 3.1 e 3.1.14. Ciascun compito principale viene analizzato sino al livello in cui compare una serie di *compiti inscindibili*, compiti per i quali non ha senso proprio una suddivisione in fasi e che, una volta avviati, vengono portati a termine o interamente annullati. I compiti inscindibili non vengono mai lasciati a metà.

Compilare un assegno è un compito inscindibile, scrivere in esso l'ammontare in euro non lo è. Rispondere al telefono in quanto rappresentante del servizio clienti non è un compito inscindibile, rispondere al telefono e soddisfare la richiesta del cliente è un compito inscindibile. I compiti inscindibili devono avere un senso e devono essere completati sino in fondo.

Il livello al quale un compito può dirsi inscindibile varia in relazione al compito. Il compito rappresentato da 3.1.14 potrà essere inscindibile eppure continuare a prevedere numerosi sottolivelli aggiuntivi. Il compito 3.2 può essere inscindibile e così può esserlo il compito 3.16.4. Ciò che conta non è lo schema di numerazione (che altro non è se non un metodo per delineare una gerarchia dei compiti) ma la decomposizione sino al livello inscindibile. I compiti inscindibili sono i mattoni fondamentali degli affari svolti. Due compiti possono ancora dirsi inscindibili anche se uno dipende occasionalmente dall'altro, ma solo se ciascuno può essere, e viene portato

a termine indipendentemente. Se due compiti dipendono sempre l'uno dall'altro, tali compiti non sono inscindibili. Il vero compito inscindibile li comprende ambedue.

Nella maggioranza degli affari si scopre rapidamente che numerosi compiti non rientrano perfettamente in uno solo dei compiti principali (livello zero), ma sembrano allargarsi a due o più compiti principali e funzionano in modo reticolare o “trasversale”. Tale situazione è praticamente sempre la prova dell'impropria definizione dei compiti principali o dell'incompleta inscindibilità dei compiti di livello più basso. L'obiettivo è quello di trasformare ciascun compito in un “oggetto” concettuale, con un'idea ben definita di cosa fa (il suo scopo nella vita) e di quali risorse (dati, calcolo, pensiero, carta, penna e via dicendo) utilizza per raggiungere i propri scopi.

### Nozioni ricavate dal documento dei dati

Dal documento dei dati è possibile ricavare numerose nozioni. In primo luogo, dato che il documento dei dati è orientato ai compiti invece che ai dati, muterà probabilmente in modo sostanziale il modo di progettazione delle finestre per l'utente. Il documento influenza sulla raccolta dei dati, sul modo in cui essi vengono presentati, su come vengono implementati gli aiuti e sul modo in cui gli utenti passano da un compito a un altro. L'orientamento ai compiti aiuta ad assicurarsi che i tipi più comuni di passaggio da un compito a un altro non richiedano da parte dell'utente sforzi al di fuori dell'ordinario.

In secondo luogo, la divisione in categorie dei compiti principali cambia via via che vengono scoperti eventuali conflitti: ciò influenza la comprensione degli affari trattati, sia da parte di chi progetta, sia da parte dell'utente.

In terzo luogo, con tutta probabilità cambia persino la frase riassuntiva. La razionalizzazione di un settore commerciale in “oggetti” relativi a compiti inscindibili causa la rimozione forzata dei retaggi del passato, dei concetti errati e delle dipendenze non strettamente necessarie che hanno a lungo appesantito senza costrutto gli affari trattati.

Il processo non è indolore, ma i benefici in termini di comprensione degli affari, di pulizia delle procedure e di automazione dei compiti superano normalmente di gran lunga i costi in termini di difficoltà emotive e di tempo investito. Se è presente una generale comprensione indirizzata allo sviluppo del progetto, è di enorme aiuto che le domande scabrose vengano poste, i presupposti errati vengano corretti e che al documento dei compiti vengano apportati aggiustamenti passo dopo passo sino a che questo sia completo.

## 2.9 Comprensione dei dati

Insieme alla scomposizione e alla descrizione dei compiti, nel documento dei compiti vengono descritte le risorse necessarie per ogni passaggio. Questa descrizione avviene compito per compito: nel documento dei dati vengono quindi inclusi i dati necessari. Questo approccio è concettualmente differente dalla visione classica dei dati. Non è più sufficiente prendere semplicemente i moduli e le maschere utilizzati da ciascun modulo applicativo e limitarsi a registrare gli elementi in esso contenuti. Il neo in questo tipo di approccio consiste nella tendenza (anche se non lo si ammette volentieri) ad accettare qualsiasi cosa come vera se stampata su carta.

Osservando ciascun compito, è necessario determinare quali dati siano necessari per svolgerlo, piuttosto che quali siano gli elementi relativi ai dati presenti nel modulo utilizzato, necessari per svolgere il compito stesso. Esigendo che la definizione dei dati discenda dal compito invece che da un qualsiasi modulo o maschera esistente, ci si obbliga a un esame del vero scopo del compito e alla reale necessità di dati. Se la persona che svolge il compito non conosce l'utili-

lizzo che viene fatto del dato inserito, tale elemento dev'essere annotato nell'elenco dei problemi da risolvere. Questo procedimento elimina una quantità enorme di cose non necessarie.

Una volta identificati, gli elementi relativi ai dati devono essere esaminati accuratamente. I codici numerici e alfabetici sono sempre sospetti. Essi nascondono vere informazioni dietro simboli non intuitivi e senza significato proprio. Esistono momenti e compiti in cui i codici sono comodi, facili da ricordare o resi necessari da grandi volumi. Ma nella progettazione finale questi casi devono essere rari e ovvi. Se così non è, significa che si è smarrita la dritta via.

Nel vaglio degli elementi relativi ai dati esistenti, i codici devono essere oggetto di una particolare attenzione. In ciascun caso è necessario domandarsi perché tale dato debba essere espresso con un codice. Il continuo utilizzo del dato sotto forma di codice dev'essere visto con sospetto. Per mantenere la codifica devono esserci buoni motivi e ragioni di forza maggiore. Il procedimento per convertire i dati codificati nella lingua locale è piuttosto semplice, ma è frutto di uno sforzo congiunto. I codici vengono in primo luogo elencati, per elemento di dato, insieme al loro significato. I codici vengono quindi esaminati da utenti e progettisti, e brevi versioni in italiano dei significati vengono proposte, discusse e approvate in via provvisoria.

Durante la stessa discussione, i progettisti e gli utenti finali devono decidere i nomi degli elementi informativi. Tali nomi sono destinati a diventare nomi di colonne nel database e a essere utilizzati regolarmente nelle query. I nomi devono pertanto essere descrittivi (evitando le abbreviazioni, a parte quelle comuni in ambito commerciale) e singolari. A causa dell'intima relazione tra il nome della colonna e i dati in questa contenuti, i due nomi devono essere specificati contemporaneamente. La scelta ragionata del nome di una colonna semplifica di gran lunga la determinazione del nome ai suoi nuovi contenuti in lingua.

Devono essere rigorosamente esaminati anche i dati non rappresentati da codici. A partire da ciò ci sono ottime ragioni per credere che tutti gli elementi di dati identificati siano necessari per le operazioni commerciali da svolgere, ma che essi non siano necessariamente ben organizzati. Ciò che appare essere un elemento di dati nel compito esistente può in effetti rivelarsi come un gruppo di più elementi mescolati tra di loro e che richiedano di essere separati. I nomi, gli indirizzi e i numeri di telefono sono esempi molto comuni di tale situazione, ma in ogni applicazione se ne possono trovare innumerevoli altri.

Nomi e cognomi sono stati mischiati, per esempio, nella tabella AUTORE. La tabella NomeAutore conteneva sia il nome, sia il cognome, anche se le tabelle erano nella Terza Forma Normale. Questo sarebbe stato un modo estremamente farraginoso per implementare realmente un'applicazione, malgrado il fatto che le regole di normalizzazione venissero tecnicamente rispettate. Per rendere un'applicazione pratica e preparare l'applicazione stessa per query in italiano, la colonna NomeAutore dev'essere scomposta in almeno due colonne nuove, Cognome e Nome. Lo stesso procedimento di suddivisione in categorie è regolarmente necessario per la razionalizzazione di altri elementi di dati ed è spesso decisamente indipendente dalla normalizzazione.

Il grado di scomposizione dipende da come i particolari elementi di dati vengono probabilmente utilizzati. È possibile spingersi troppo avanti e operare la suddivisione in categorie che, per quanto formate da pezzi separabili, non danno nessun ulteriore vantaggio nel loro nuovo stato. La suddivisione dipende dall'applicazione e dev'essere valutata elemento per elemento. Una volta terminata la suddivisione, ai nuovi elementi creati, destinati a diventare colonne, dev'essere assegnato un nome adeguato, e i dati che essi contengono devono essere esaminati. I dati di testo che ricadano entro un numero definito di valori devono essere revisionati per l'assegnazione di un nome. Tali nomi di colonne e valori, come quelli dei codici, risultano essere provvisori.

## I modelli di dati inscindibili

Ha ora inizio il procedimento di normalizzazione, e con esso la definizione dei modelli di dati inscindibili. Esistono molti testi validi sull'argomento e una vasta gamma di strumenti di analisi e progettazione in grado di accelerare il processo. Per questo motivo non viene suggerito alcun metodo particolare in queste pagine, in quanto raccomandando un metodo si rischia di metterne in ombra un altro.

Ciascuna transazione inscindibile dev'essere modellata e le dovrà essere assegnato, come etichetta, il numero del compito a cui si riferisce. Nel modello sono compresi i nomi delle tabelle, le chiavi primarie ed estranee e le colonne principali. Ciascuna relazione normalizzata deve avere un nome descrittivo e con ciascuna tabella deve comparire un calcolo stimato delle righe e delle percentuali di transazioni. Ogni modello è accompagnato, facoltativamente, da un foglio aggiuntivo con l'elenco di tutte le colonne e i tipi di dati, i relativi intervalli di valori e i nomi provvisori per le tabelle, le colonne e i valori nelle colonne a cui è assegnato un nome.

## Il modello commerciale inscindibile

Il documento dei dati viene combinato con il documento dei compiti. Il documento combinato è un modello commerciale. Tale documento viene revisionato congiuntamente da chi progetta l'applicazione e dagli utenti finali, allo scopo di verificarne l'accuratezza e la completezza.

## Il modello commerciale

A questo punto, sia chi progetta l'applicazione, sia gli utenti finali dovrebbero avere una chiara visione degli affari trattati, dei relativi compiti da svolgere e dei dati utilizzati. Una volta corretto e approvato il modello commerciale, il processo di sintesi dei compiti e dei modelli di dati in un modello commerciale complessivo ha inizio. Questa parte del procedimento ordina gli elementi di dati comuni a più compiti, completa la normalizzazione finale su larga scala e individua nomi coerenti e definitivi per tutte le parti.

Tutto ciò può essere un'operazione di portata decisamente vasta per le applicazioni principali, con documentazione di supporto che comprende i compiti, i modelli di dati (con nomi corretti per gli elementi, basati sul modello completo) e un elenco di ciascuna delle tabelle a scala intera e dei relativi nomi delle colonne, tipi di dati e contenuti. La verifica finale degli sforzi prodigati avviene seguendo il percorso di accesso ai dati di ciascuna transazione nel modello commerciale completo, per fare in modo che tutti i dati richiesti per la transazione siano disponibili per la selezione o l'inserimento e che nessun compito preveda l'inserimento di dati essenziali per l'integrità referenziale del modello, con elementi mancanti.

Fatta eccezione per gli sforzi devoluti nell'individuazione di nomi adeguati per le varie tabelle e colonne e per i valori comuni, praticamente tutto quanto fatto sino a questo punto fa parte dell'analisi e non della progettazione. L'obiettivo è la promozione della comprensione degli affari trattati e delle relative componenti.

## Inserimento dei dati

La progettazione delle finestre di inserimento dei dati non deriva dal modello commerciale. Essa non si concentra sulle tabelle, ma piuttosto sui compiti. Per tale motivo vengono create finestre che supportino l'orientamento al compito e la necessità di passare da un compito secondario a un altro, se necessario. In termini pratici, tutto ciò comporta spesso l'utilizzo di una tabella princi-

pale utilizzata dal compito, e di altre tabelle sulle quali possano essere eseguite query per ricavare i valori cercati o che potranno essere aggiornate in caso di accesso alla tabella principale.

Vi sono però anche occasioni in cui semplicemente non esiste una tabella principale, ma al suo posto sono presenti numerose tabelle correlate, ciascuna delle quali fornisce o riceve dati a supporto del compito svolto. Tali finestre appaiono e si comportano in modo decisamente diverso da quelle tipiche orientate alla tabella sviluppate per molte applicazioni, ma ampliano in modo significativo l'efficienza del lavoro dei propri utenti e i relativi contributi alle operazioni commerciali. E questo è precisamente l'unico scopo del nostro approccio.

L'interazione tra utente e macchina è fondamentale, le maschere di inserimento e le query devono essere coerentemente orientate ai compiti e descrittive, in lingua corrente. Anche l'utilizzo di icone e interfacce grafiche gioca un ruolo importante. Le maschere devono riflettere il modo con cui il lavoro viene effettivamente portato a termine ed essere costruite in modo da rispondere nella lingua in cui gli affari vengono trattati.

## Query e report

Se c'è qualcosa che distingue l'approccio relazionale e il linguaggio SQL da ambienti applicativi più tradizionali, è la possibilità per gli utenti finali di imparare agevolmente ed eseguire query ad hoc. Si tratta di report e di query occasionali che non fanno parte del gruppo di base normalmente sviluppato e fornito insieme al codice dell'applicazione.

Con SQLPLUS (e altri strumenti di report), gli utenti finali ottengono un controllo senza precedenti sui propri dati. Sia gli utenti, sia gli sviluppatori beneficiano di tale possibilità: gli utenti perché vengono messi in grado di costruire dei report, analizzare le informazioni, modificare le query ed eseguirle nuovamente nel giro di pochi minuti; gli sviluppatori in quanto vengono sollevati dall'indesiderato compito di creare nuovi report.

Agli utenti viene consentito di osservare in profondità i propri dati, analizzarli e rispondere con una velocità e un'immediatezza inimmaginabile solo pochi anni or sono. Questo salto di produttività viene grandemente aumentato se le tabelle, le colonne e i valori dei dati sono scrupolosamente riportati in italiano, mentre viene enormemente pregiudicato se si permette a cattive convenzioni per l'assegnazione dei nomi e a codici e abbreviazioni senza senso di inficiare il progetto. Il tempo impiegato nel procedimento del progetto per assegnare agli oggetti nomi coerenti e descrittivi in italiano viene presto ricompensato dagli utenti e di conseguenza dall'andamento degli affari.

Alcuni, normalmente chi non ha mai realizzato applicazioni relazionali di grandi dimensioni, temono che il passaggio dei dispositivi di query agli utenti finali possa sovraccaricare la macchina su cui tali dispositivi vengono utilizzati. Si teme che gli utenti scrivano query inefficaci che consumano un numero esorbitante di cicli della CPU, rallentando la macchina e tutti gli altri utenti. L'esperienza mostra che ciò non è sempre vero. Gli utenti imparano rapidamente a capire quali tipi di query funzionano rapidamente e quali no. Inoltre, la maggior parte degli strumenti di business intelligence e di generazione di report attualmente disponibili sono in grado di stimare la quantità di tempo necessaria per l'esecuzione di una query e di limitare l'accesso, in base all'utente, all'ora del giorno o ad entrambi i criteri, alle query che consumerebbero una quantità sproporzionata di risorse. In pratica, le fatiche imposte dagli utenti a una macchina sfuggono loro di mano solo occasionalmente, mentre i benefici ottenuti superano di gran lunga i costi in termini di elaborazione. Praticamente ogniqualvolta sia possibile spostare uno sforzo da una persona a una macchina, si risparmia denaro.

Lo scopo effettivo del progetto è quello di rendere chiare e soddisfare le esigenze commerciali e degli utenti. Se deve esistere un'imparzialità, questa dovrà sempre essere orientata a rendere l'applicazione più semplice da comprendere e utilizzare, in particolare a spese della CPU o

del disco, ma in misura minore se il prezzo di ciò è una complessità interna tanto grande che la manutenzione e le modifiche diventano difficoltose e lente.

## 2.10 Verso la normalizzazione dei nomi degli oggetti

L'approccio di base consiste nella scelta di nomi significativi, facili da memorizzare e descrittivi, evitando le abbreviazioni e i codici e utilizzando il trattino di sottolineatura in modo coerente, oppure non utilizzandolo affatto. In un'applicazione di grandi dimensioni, i nomi delle tabelle, delle colonne e dei dati sono spesso formati da più parole, come ad esempio CreditiContestatiAddebitati, oppure Data\_Ultima\_Chiusura\_GL. L'obiettivo dei metodi di nomenclatura meditati è la facilità di utilizzo: i nomi devono essere facilmente memorizzabili e devono seguire regole facili da spiegare e da applicare. Nelle pagine che seguono verrà presentato un approccio più rigoroso per quanto riguarda la nomenclatura, allo scopo di sviluppare un processo formale di normalizzazione dei nomi degli oggetti.

### Integrità a livello dei nomi

In un database relazionale la gerarchia degli oggetti spazia dal database ai proprietari delle tabelle, alle tabelle, alle colonne, ai valori dei dati. In sistemi molto grandi possono persino esservi più database e questi possono essere distribuiti all'interno di varie locazioni. Per brevità i livelli più alti vengono per ora ignorati, ma quanto verrà detto può essere applicato anche a essi.

Ciascun livello di questa gerarchia è definito all'interno del livello superiore e a ciascuno devono essere assegnati nomi appropriati e non incorporati dall'esterno. Per esempio, una tabella non può avere due colonne denominate Nome, e l'account di nome George non può essere proprietario di due tabelle denominate AUTORE.

Non vi è ragione per cui ciascuna tabella di George debba avere un nome unico nell'intero database. Inoltre, altri proprietari possono possedere tabelle di nome AUTORE. Anche se a George è garantito l'accesso a queste tabelle, non vi è confusione, poiché è possibile identificarle in modo univoco anteponendo al nome della tabella il prefisso del nome del proprietario, come per Dietrich.AUTORE. Non sarebbe coerente da un punto di vista logico incorporare il nome del proprietario George nel nome di ciascuna delle sue tabelle, come in AUTOREGEORGE, BIBLIOTECAGEORGE e così via. Inserendo parte del nome del livello superiore nei livelli di proprietà di quest'ultimo non si ottiene altro effetto che rendere i nomi delle tabelle più confusi e complicati, operando a tutti gli effetti una violazione dell'*integrità a livello del nome*.

La concisione non dovrebbe mai essere privilegiata rispetto alla chiarezza. L'inserimento di parti di nomi della tabella nei nomi delle colonne è una tecnica da evitare perché viola l'idea logica dei livelli e l'integrità a livello del nome da essa richiesta. Inoltre crea confusione, richiedendo agli utenti di cercare i nomi delle colonne ogni volta che vogliono scrivere una query. I nomi degli oggetti devono essere unici all'interno del livello a essi gerarchicamente superiore, ma non dovrebbe essere ammessa alcuna incorporazione di nomi da un livello esterno a quello dell'oggetto stesso.

Il supporto per i tipi di dati astratti in Oracle rafforza la capacità di creare nomi coerenti per gli attributi. Se si crea un tipo di dato chiamato INDIRIZZO\_TY, esso avrà gli stessi attributi ogni volta che viene impiegato. Ciascuno degli attributi avrà un nome, un tipo di dato e una lunghezza coerenti, rendendo l'implementazione più congruente in tutta l'impresa. Tuttavia l'uso dei tipi di dati astratti in questo modo implica l'adozione delle seguenti pratiche:

- una corretta definizione dei tipi di dati sin dall'inizio, in modo da evitare la necessità di modificare un tipo di dato in un secondo momento;
- supporto dei requisiti di sintassi dei tipi di dati astratti (consultare il Capitolo 4 per ulteriori dettagli).

## Chiavi esterne

Una difficoltà dell'uso di nomi di colonna brevi è la comparsa occasionale di una chiave esterna in una tabella in cui vi sia un'altra colonna che ha lo stesso nome della chiave esterna nella propria tabella di base. Una possibile soluzione a lungo termine consiste nel consentire l'utilizzo del nome completo della chiave esterna, incluso il nome della tabella di base di quest'ultima, come nome della colonna nella tabella locale (come per esempio BIBLIOTECA.Titolo come nome di colonna).

La necessità pratica di risolvere i problemi legati a colonne con lo stesso nome richiede di intraprendere una delle azioni seguenti.

- Inventare un nome che comprenda la tabella sorgente della chiave esterna senza utilizzare il punto (per esempio ricorrendo al trattino di sottolineatura).
- Inventare un nome che incorpori un'abbreviazione della tabella sorgente della chiave esterna.
- Inventare un nome differente da quello utilizzato nella tabella sorgente.
- Cambiare il nome della colonna che crea conflitto.

Nessuna delle soluzioni precedenti è particolarmente allettante, ma se si incorre in un dilemma di nomi identici, è necessario adottare una di queste soluzioni.

## Nomi singolari

Un'area di grande confusione è rappresentata dalla domanda se gli oggetti debbano avere nomi singolari o plurali. Tabella AUTORE o AUTORI? Colonna Nome o Nomi?

Esistono due modalità che vengono in aiuto nella risoluzione di questo problema. Per prima cosa si considerino alcune colonne comuni a quasi ogni database: Nome, Indirizzo, Città, Provincia, Cap. A parte la prima, a qualcuno è mai capitato di vedere i nomi di queste colonne scritti al plurale? È praticamente scontato, quando si considerano i nomi di queste colonne, che essi descrivano il contenuto di un'unica riga, un record. Anche se i database relazionali sono “orientati agli insiemi”, è chiaro che l’unità fondamentale di un insieme è una riga ed è al contenuto di questa riga che fanno riferimento i nomi singolari delle colonne. Il progetto di una schermata di inserimento dei dati per catturare il nome e l’indirizzo di una persona potrebbe essere simile al seguente?

Nomi: \_\_\_\_\_

Indirizzi: \_\_\_\_\_

Città: \_\_\_\_\_ Province \_\_\_\_ CAP \_\_\_\_

Oppure i nomi delle colonne vengono scritti al singolare sullo schermo, poiché si riferiscono al nome e all’indirizzo di *una* persona alla volta, mentre occorre avvisare l’utente che quando scrive le query deve convertirli al plurale? È sicuramente molto più intuitivo e lineare specificare i nomi delle colonne al singolare.

Se tutti gli oggetti sono stati denominati in modo coerente, né il progettista né l'utente deve cercare di ricordare le regole secondo cui utilizzare il plurale oppure no. Il vantaggio dovrebbe essere ovvio. Si supponga di decidere che d'ora in poi tutti gli oggetti saranno plurali. Si avranno "i", o "e" come finali del nome di ciascun oggetto, magari anche alla fine di ciascuna parola di nomi lunghi composti da più parole. Che vantaggi può portare questa scelta? Tutto ciò è facile da utilizzare, da capire, da ricordare? Ovviamente no!

La soluzione migliore pertanto è la seguente: tutti i nomi di oggetti sono sempre singolari. L'unica eccezione a questa regola è rappresentata da qualunque termine ampiamente accettato e già comunemente utilizzato nel mondo degli affari, come "vendite".

## Brevità

Come si è già detto in precedenza, la chiarezza non dovrebbe mai essere sacrificata a beneficio della concisione, ma dati due nomi ugualmente significativi, mnemonici e descrittivi, conviene sempre scegliere il più breve. Durante lo sviluppo delle applicazioni, conviene proporre nomi di colonne e tabelle alternativi come quelli presentati al gruppo di utenti e sviluppatori e ascoltare le loro proposte per quanto riguarda la scelta del nome più chiaro. Come si fa a costruire un elenco di alternative? Si può utilizzare un dizionario normale e uno dei sinonimi. In un gruppo di lavoro dedicato a sviluppare applicazioni produttive superiori, a ciascun membro dovrebbe essere fornito un dizionario e un dizionario dei sinonimi e ricordata più volte l'importanza di assegnare con cura i nomi agli oggetti.

## Sinonimi di nomi di oggetti

I database relazionali dovrebbero includere un dizionario dei sinonimi per i nomi degli oggetti, proprio come avviene per il dizionario dei dati. Questo dizionario dovrebbe far osservare gli standard di nomenclatura della società e assicurare coerenza nella scelta dei nomi e delle abbreviazioni (dove utilizzate).

Tali standard possono richiedere l'utilizzo del trattino di sottolineatura in modo da rendere più semplice l'analisi delle componenti del nome. In questo modo viene inoltre forzato l'utilizzo coerente dei trattini di sottolineatura, mentre al momento se ne fa un uso disperso e inconsistente.

Se si lavora direttamente con un'agenzia governativa o con un'importante società, queste ultime potrebbero già aver definito dei propri standard per la denominazione degli oggetti. Nel corso degli anni gli standard di denominazione degli oggetti delle grandi società si sono irradiati al resto del mercato commerciale e possono costituire la base per la definizione degli standard di denominazione impiegati dalla propria società. Per esempio, questi standard potrebbero fornire direttive relative alla scelta tra i termini "Società" e "Ditta". In caso contrario, si dovrebbero sviluppare degli standard di denominazione che siano coerenti sia con gli standard di base sia con le linee guida offerte in questo capitolo.

### 2.11 Chiavi intelligenti e valori di colonna

Le chiavi *intelligenti* vengono così chiamate perché contengono combinazioni significative di informazioni. Il termine è assai ingannevole in quanto implica qualcosa di positivo o degno di merito. Un termine più idoneo potrebbe essere chiavi "sovrafficate". I registri generali e i

codici dei prodotti spesso rientrano in questa categoria e comprendono tutte le difficoltà associate agli altri codici e altro. Inoltre, le difficoltà che si riscontrano con le chiavi sovraccaricate si applicano anche a colonne non chiave costruite con più di una porzione di dati significativi.

Una descrizione tipica di una chiave sovraccarica o di un valore di colonna è la seguente: "Il primo carattere rappresenta il codice della regione. I successivi quattro caratteri rappresentano il numero del catalogo. La cifra finale è il codice del costo centrale, a meno che questo non sia una parte importata, nel qual caso una 'T' viene accodata al numero, o a meno che si tratti di un elemento voluminoso, per esempio viti, in cui vengono utilizzate solo tre cifre per il numero del catalogo e il codice della regione è HD".

L'eliminazione delle chiavi sovraccaricate e dei valori delle colonne è essenziale in un buon progetto di tipo relazionale. Le dipendenze costruite su parti di queste chiavi (solitamente chiavi esterne in altre tabella) sono tutte a rischio nel caso di manutenzione della struttura. Sfortunatamente, molte aree applicative possiedono chiavi sovraccaricate che sono state utilizzate per anni e sono profondamente inglobate nell'attività della società. Alcune di esse vennero create durante i primi tentativi di automazione, con database che non erano in grado di gestire colonne a più chiavi per chiavi composte. Altre derivano da un retaggio storico, che utilizza codici brevi, solitamente numerici, per gestire più significati e coprire più casi di quanti non ne fossero inizialmente previsti. L'eliminazione di chiavi sovraccaricate esistenti può avere conseguenze pratiche che ne rendono impossibile l'attuazione immediata. Ciò rende più difficile la costruzione di una nuova applicazione relazionale.

La soluzione a questo problema consiste nel creare un nuovo gruppo di chiavi, sia primarie sia esterne, che normalizzino in modo appropriato i dati; successivamente è necessario assicurarsi che gli utenti possano accedere alle tabelle solo attraverso queste chiavi. La chiave sovraccarica viene quindi mantenuta come una colonna aggiuntiva e unica della tabella. L'accesso a quest'ultima è ancora possibile utilizzando i metodi passati (per esempio trovando le corrispondenze della chiave sovraccarica in una query), ma le chiavi con la nuova struttura vengono suggerite come metodo preferenziale di accesso ai dati. Nel tempo, con un addestramento opportuno, gli utenti si rivolgeranno verso queste nuove chiavi. Alla fine, le chiavi sovraccaricate (e gli altri valori di colonna sovraccarichi) potranno semplicemente essere annullate (impostandole a NULL) o rimosse dalla tabella.

Se non si riescono a eliminare chiavi e valori sovraccarichi, si incrementa notevolmente il costo e la difficoltà di operazioni quali l'estrazione di informazioni dal database, la convalida dei valori, l'integrità dei dati e la modifica della struttura.

## 2.12 Il decalogo

A questo punto tutti i principali problemi legati alla progettazioni di applicazioni destinate a un ambiente produttivo sono stati analizzati. Probabilmente è utile riassumere gli aspetti affrontati in un unico elenco, ed ecco così "Il decalogo". La loro presentazione non ha alcun intento di dire al lettore ciò che dovrebbe fare, ma piuttosto dà per scontato che egli sia in grado di apportare modifiche razionali e possa beneficiare dell'esperienza di altri che hanno affrontato le stesse sfide. Lo scopo non è quello di descrivere il ciclo di sviluppo, che il lettore probabilmente conosce già fin troppo bene, ma piuttosto quello di influenzare questo sviluppo con un orientamento che modifichi radicalmente il modo in cui appare e viene utilizzata l'applicazione. Se si tengono nella dovuta considerazione queste idee, è possibile migliorare notevolmente la produttività e l'umore degli utenti di un'applicazione.

## I dieci comandamenti della progettazione

1. Far partecipare gli utenti. Inserirli nel gruppo di progettazione e insegnare loro il modello relazionale e SQL.
2. Scegliere i nomi delle tabelle, delle colonne e delle chiavi assieme agli utenti. Sviluppare un dizionario dei sinonimi per l'applicazione in modo da assicurare nomi coerenti.
3. Utilizzare parole significative, facilmente memorizzabili, descrittive, brevi e al singolare. Se si utilizzano i trattini di sottolineatura, lo si faccia coerentemente, oppure non li si utilizzi affatto.
4. Non mescolare tra loro livelli differenti all'interno dei nomi.
5. Evitare l'utilizzo di codici e abbreviazioni.
6. Utilizzare chiavi significative dove possibile.
7. Scomporre le chiavi sovraccaricate.
8. Effettuare l'analisi e la progettazione tenendo conto delle operazioni richieste, non solamente dei dati. Tenere presente che la normalizzazione non fa parte della progettazione.
9. Spostare le operazioni da svolgere dagli utenti alla macchina. È utile spendere di più in cicli e operazioni di memorizzazione, se si ottiene una maggiore facilità di utilizzo.
10. Non farsi tentare dalla velocità di sviluppo. Dedicare tempo e attenzione all'analisi, alla progettazione, alle prove e alla messa a punto.

## Capitolo 3

# Le parti fondamentali del discorso in SQL

- 3.1 **Stile**
- 3.2 **Creazione della tabella GIORNALE**
- 3.3 **Uso di SQL per selezionare dati da tabelle**
- 3.4 **I comandi select, from, where e order by**
- 3.5 **Logica e valore**
- 3.6 **Un altro impiego delle subquery con where**
- 3.7 **Combinazione di tabelle**
- 3.8 **Creazione di una vista**

**C**on SQL (Structured Query Language), è possibile conversare con Oracle chiedendo le informazioni che si desidera selezionare (select), inserire (insert), aggiornare (update) o cancellare (delete). In effetti, questi sono i quattro verbi fondamentali che si utilizzano per fornire istruzioni a Oracle. A partire da Oracle9i è inoltre possibile utilizzare un comando aggiuntivo, merge, per eseguire operazioni di inserimento e aggiornamento con un singolo comando.

Nel Capitolo 1 si è visto che cosa si intende con il termine “relazionale”, come vengono organizzate le tabelle in colonne e righe e come dare istruzioni a Oracle per selezionare determinate colonne da una tabella e visualizzare le informazioni contenute riga dopo riga. In questo capitolo e nei seguenti viene illustrata in maniera più completa l’esecuzione di queste procedure per i diversi tipi di dati supportati da Oracle. In questo paragrafo viene spiegato come interagire con SQL\*PLUS, un prodotto molto potente di Oracle che consente di raccogliere le istruzioni fornite dall’utente, controllarne la correttezza e quindi sottoporle a Oracle, e successivamente modificare o riformulare la risposta fornita da Oracle sulla base degli ordini o delle direttive impartiti. Questo è uno strumento con il quale è possibile *interagire*, ovvero l’utente può “parlare” con esso e ricevere “risposte”. Si possono impartire direttive che vengono seguite in maniera precisa. Se le istruzioni non risultano comprensibili, viene visualizzato un messaggio di segnalazione.

All’inizio, comprendere la differenza tra ciò che viene eseguito da SQL\*PLUS e ciò che viene eseguito da Oracle può apparire un po’ complicato, soprattutto perché i messaggi di errore prodotti da Oracle vengono semplicemente passati all’utente da SQL\*PLUS; tuttavia, proseguendo con la lettura di questo testo le differenze risulteranno più evidenti. Nelle fasi di apprendimento, si può pensare a SQL\*PLUS come a un semplice collaboratore, un assistente che segue le istruzioni dell’utente e lo aiuta a eseguire il lavoro in maniera più veloce. Si interagisce con questo collaboratore semplicemente digitando con la tastiera.

Si possono seguire gli esempi proposti in questo capitolo e nei seguenti semplicemente digitando i comandi illustrati, ottenendo dai programmi Oracle e SQL\*PLUS esattamente le stesse risposte. Occorre soltanto assicurarsi che le tabelle utilizzate in questo testo siano state caricate nella propria copia di Oracle.

In ogni caso è possibile comprendere le procedure descritte anche senza effettivamente metterle in pratica; per esempio, è possibile utilizzare i comandi illustrati sulle proprie tabelle. Tuttavia, tutto risulterà probabilmente più chiaro e più semplice, se si utilizzano in Oracle le stesse tabelle proposte in questo testo e ci si esercita impostando le stesse query.

Il CD-ROM allegato al libro contiene le istruzioni per caricare tali tabelle. Presupponendo che ciò sia già stato fatto, occorre connettersi a SQL\*PLUS e iniziare a lavorare digitando:

```
sqlplus
```

(Se si desidera eseguire SQL\*PLUS dal proprio computer desktop client, occorre selezionare l'opzione di programma SQL Plus dall'opzione di menu Application Development sotto l'opzione di menu del software Oracle). Così facendo viene avviato SQL\*PLUS (si osservi che non deve essere digitato l'asterisco \* che si trova a metà della denominazione ufficiale del prodotto e che l'asterisco non appare nemmeno nel nome del programma. Da questo punto in poi, si fa riferimento a SQLPLUS senza l'asterisco). Dal momento che in Oracle sono previsti accurati strumenti di protezione dei dati memorizzati, ogni volta che ci si connette è necessario immettere un ID e una password. Viene visualizzato un messaggio di copyright e quindi la richiesta di nome utente e password. Occorre connettersi al database utilizzando l'account e la password creati per contenere le tabelle di esempio. Quando si forniscono un nome utente e una password validi, viene visualizzato un messaggio in cui si segnala che l'utente è connesso a Oracle e quindi il seguente prompt:

```
SQL>
```

Ora SQLPLUS è attivo, in attesa di istruzioni. Se il comando fallisce, le ragioni possibili sono diverse: Oracle non è nel percorso indicato, non si dispone dell'autorizzazione necessaria per l'esecuzione di SQLPLUS, oppure Oracle non è stato installato in maniera appropriata. Se appare questo messaggio:

```
ERROR: ORA-1017: invalid username/password; logon denied
```

può significare che sono stati inseriti in maniera errata il nome utente o la password, oppure che il proprio nome utente non è stato ancora impostato sulla copia di Oracle in uso. Dopo tre tentativi di inserimento di nome utente o di password falliti, il tentativo di connessione viene bloccato da SQLPLUS con questo messaggio:

```
unable to CONNECT to ORACLE after 3 attempts, exiting SQL*Plus
```

Se viene visualizzato questo messaggio, occorre contattare l'amministratore di database dell'azienda. Se tutto funziona e viene visualizzato il prompt SQL>, si può iniziare a lavorare con SQLPLUS.

Quando si desidera interrompere il lavoro e uscire da SQLPLUS, è sufficiente digitare:

```
quit
```

### 3.1 Stile

Innanzitutto, alcune osservazioni sullo stile. Per SQLPLUS non è importante se i comandi SQL vengono digitati in maiuscolo o minuscolo. Con questo comando:

SeLeCt argoMENTO, sezione, PAGINA FROM gioRNAlE;

si ottiene esattamente lo stesso risultato rispetto al comando:

select Argomento, Sezione, Pagina from GIORNALE;

L'utilizzo di lettere maiuscole o minuscole è importante soltanto quando in SQLPLUS o in Oracle viene verificata l'uguaglianza di valori alfanumerici. Se si indica a Oracle di trovare una riga in cui Sezione = 'f' e Sezione è in realtà corrispondente a 'F', Oracle non trova nulla (perché f e F non coincidono). Tranne che in questo tipo di utilizzo, l'impiego di lettere maiuscole e minuscole è del tutto irrilevante. Per inciso, la lettera 'F', in questo uso, viene definita come *elemento letterale*, ovvero significa che nella colonna Sezione deve essere verificata la presenza della lettera 'F' e non ricercata una colonna denominata F. Gli apici singoli che racchiudono la lettera indicano che si tratta di un letterale e non di un nome di colonna.

Come scelta stilistica, in questo testo vengono seguite alcune convenzioni sul formato delle lettere per rendere il testo più agevole da leggere.

- I termini select, from, where, order by, having e group by vengono sempre indicati in minuscolo e con un tipo di carattere diverso da quello del corpo del testo.
- Anche i comandi SQLPLUS sono riportati nello stesso stile: column, set, save, ttitle e così via.
- IN, BETWEEN, UPPER e altri operatori e funzioni di SQL vengono riportati in maiuscolo e con un tipo di carattere diverso.
- I nomi di colonna sono riportati con l'iniziale maiuscola e il tipo di carattere normale: Argomento, EstOvest, Longitudine e così via.
- Per i nomi di tabelle vengono utilizzati caratteri maiuscoli con il tipo di carattere normale: GIORNALE, CLIMA, DISLOCAZIONE e così via.

Si possono seguire convenzioni analoghe per la creazione delle proprie query, oppure può essere che l'azienda per cui si lavora abbia degli standard già impostati; si potrebbe anche scegliere di inventare delle convenzioni proprie. In ogni caso lo scopo di questo tipo di standard dovrebbe sempre essere quello di rendere il lavoro semplice da leggere e da comprendere.

## 3.2 Creazione della tabella GIORNALE

Gli esempi di questo libro si basano sulle tabelle create tramite gli script collocati sul CD-ROM allegato. Ciascuna tabella viene creata con il comando create table, che specifica i nomi delle colonne nella tabella, nonché le caratteristiche di tali colonne. Ecco il comando create table necessario per la tabella GIORNALE, che sarà utilizzata in molti degli esempi di questo libro:

```
create table GIORNALE (
Argomento  VARCHAR2(15) not null,
Sezione    CHAR(1),
Pagina     NUMBER
);
```

Nei capitoli successivi del libro si vedrà come interpretare tutte le clausole di questo comando. Per il momento lo si può leggere nel modo seguente: "Creare una tabella chiamata GIORNALE, che avrà tre colonne denominate Argomento (una colonna di caratteri a lunghezza variabile), Sezione (una colonna di caratteri a lunghezza fissa) e Pagina (una colonna numerica). I valori della colonna Argomento possono avere una lunghezza massima di 15 caratteri e tutte le

righe devono avere un valore per Argomento. I valori di Sezione avranno tutti lunghezza pari a 1 carattere”.

Nei capitoli successivi si vedrà come estendere questo semplice comando in modo da aggiungere vincoli, indici e storage clause (clausole di memorizzazione). Per il momento, la tabella GIORNALE verrà mantenuta il più possibile semplice in modo che gli esempi possano concentrarsi sul linguaggio SQL.

### 3.3 Uso di SQL per selezionare dati da tabelle

Nella Figura 3.1 è visualizzata una tabella di argomenti tratta da un giornale locale. Se fosse una tabella Oracle, invece che semplice carta e inchiostro sulla facciata del giornale locale, verrebbe visualizzata da SQLPLUS digitando questo comando:

```
select Argomento, Sezione, Pagina from GIORNALE;
```

ARGOMENTO	S	PAGINA
Notizie	A	1
Sport	D	1
Editoriali	A	12
Economia	E	1
Meteo	C	2
Televisione	B	7
Nascite	F	7
Annunci	F	8
Salute	F	6
Vita moderna	B	1
Fumetti	C	4

Argomento	Sezione	Pagina
Annunci	F	8
Bridge	B	2
Economia	E	1
Editoriali	A	12
Film	B	4
Fumetti	C	4
Meteo	C	2
Nascite	F	7
Necrologi	F	6
Notizie	A	1
Salute	F	6
Sport	D	1
Televisione	B	7
Vita moderna	B	1

Figura 3.1 Una tabella denominata GIORNALE.

Film	B	4
Bridge	B	2
Necrologi	F	6

14 rows selected.

Che cosa c'è di diverso tra la tabella creata in precedenza e quella visualizzata nell'output della Figura 3.1? In entrambe sono riportate le stesse informazioni, ma il formato cambia; per esempio, le intestazioni di colonna sono leggermente diverse da quelle richiamate con il comando select.

La colonna denominata Sezione appare indicata soltanto con la lettera 'S'; inoltre, anche se sono state utilizzate lettere maiuscole e minuscole per digitare la linea di comando:

```
select Argomento, Sezione, Pagina from GIORNALE;
```

le denominazioni delle colonne vengono visualizzate con caratteri maiuscoli.

Questi cambiamenti sono il risultato delle elaborazioni svolte da SQLPLUS sulle modalità con cui le informazioni dovrebbero essere presentate. È senz'altro possibile modificare tali presupposizioni, tuttavia finché non si assegnano modalità diverse, ecco come viene trasformato da SQLPLUS ciò che l'utente digita:

- tutte le intestazioni di colonna sono riportate in caratteri maiuscoli;
- l'ampiezza delle colonne è fissata dal valore definito in Oracle;
- vengono eliminati gli spazi tra le parole, se l'intestazione di colonna rappresenta una funzione (come mostrato nel Capitolo 7).

Il primo punto risulta evidente: i nomi di colonna digitati sono visualizzati in caratteri maiuscoli. Il secondo punto non è così ovvio. Come vengono definiti i parametri per le colonne? Per scoprirlo, è sufficiente chiederlo a Oracle. È sufficiente chiedere a SQLPLUS di descrivere la tabella, come illustrato di seguito:

```
describe GIORNALE
```

Name	Null?	Type
ARGOMENTO	NOT NULL	VARCHAR2(15)
SEZIONE		CHAR(1)
PAGINA		NUMBER

Viene visualizzata una tabella descrittiva in cui sono elencate le colonne e la loro definizione per quanto concerne la tabella GIORNALE; il comando describe funziona per qualsiasi tabella. Si osservi che i dettagli di questa descrizione corrispondono al comando create table fornito in precedenza nel capitolo.

Nella prima colonna sono indicate le denominazioni delle colonne nella tabella descritta.

Nella seconda colonna, Null?, viene indicata la regola seguita per la colonna citata a sinistra. Al momento della creazione della tabella GIORNALE, la regola NOT NULL ha indicato a Oracle di non consentire ad alcun utente di aggiungere una nuova riga alla tabella nel caso in cui la colonna Argomento risulti vuota (NULL significa vuoto). Forse in una tabella come GIORNALE sarebbe stato opportuno impostare la stessa regola per tutte e tre le colonne. Che senso ha conoscere una voce di Argomento senza sapere in quale Sezione e in quale Pagina si trova? Tuttavia, per questioni di semplicità, in questo caso solo la colonna Argomento è stata impostata con la regola specifica di non poter essere vuota.

Il fatto che Sezione e Pagina non abbiano nessun parametro nella colonna Null? significa che queste colonne della tabella GIORNALE possono avere delle righe vuote.

Nella terza colonna, Type, sono contenute informazioni sulla natura delle singole colonne. Argomento è una colonna di tipo VARCHAR2 (numero di caratteri variabile) in cui possono essere inseriti fino a 15 caratteri (lettere, numeri, simboli o spazi).

Anche Sezione è una colonna in cui è possibile inserire caratteri, tuttavia può contenerne uno solo. Chi ha creato la tabella sapeva che le sezioni del giornale sono rappresentate con una singola lettera, per cui la colonna è stata definita con l'ampiezza strettamente necessaria. È stato inoltre impostato il tipo di dati CHAR, utilizzato per stringhe di caratteri di lunghezza prefissata. Quando è stato visualizzato da SQLPLUS il risultato della query impostata:

```
select Argomento, Sezione, Pagina from GIORNALE;
```

Oracle ha trasmesso l'informazione che la colonna Sezione poteva contenere al massimo un solo carattere. Si è quindi presupposto che non fosse necessario utilizzare uno spazio maggiore di quello impostato, per cui è stata visualizzata una colonna con ampiezza sufficiente per un solo carattere e il nome della colonna è stato conseguentemente ridotto alla sola iniziale: 'S'.

La terza colonna della tabella GIORNALE è Pagina e contiene semplicemente un numero. Si osservi che la colonna Pagina viene visualizzata con ampiezza pari a dieci caratteri, anche se non ci sono pagine il cui numero ha più di due cifre: per i numeri generalmente non viene definita un'ampiezza massima, per cui viene automaticamente impostata una larghezza sufficiente per iniziare.

Inoltre il margine dell'intestazione della sola colonna composta unicamente da numeri, Pagina, è allineato a destra, mentre le intestazioni delle colonne che contengono caratteri sono allineate a sinistra. Si tratta dell'allineamento delle intestazioni di colonna impostato da SQLPLUS. È possibile modificare le caratteristiche di allineamento a seconda delle necessità, come avviene con le altre caratteristiche che riguardano il formato delle colonne.

Infine, con SQLPLUS è possibile sapere quante righe sono presenti nella tabella GIORNALE di Oracle (si osservi l'annotazione "14 rows selected" nella parte sottostante la tabella GIORNALE). Questa caratteristica viene chiamata *feedback*. È possibile impostare SQLPLUS in modo da non visualizzare questi messaggi impostando l'opzione feedback nel modo illustrato di seguito:

```
set feedback off
```

oppure è possibile impostare un numero minimo di righe affinché la funzione di feedback si attivi:

```
set feedback 25
```

Con quest'ultimo esempio si indica a Oracle che non si desidera conoscere il numero di righe visualizzate se questo non è almeno 25. Se non si specifica un valore diverso, l'opzione feedback è impostata a 6.

set è un comando di SQLPLUS, ovvero si tratta di un'istruzione che indica a SQLPLUS le modalità da seguire. Sono molte le caratteristiche di SQLPLUS che è possibile impostare, come feedback; diverse vengono illustrate e utilizzate in questo capitolo e in quelli seguenti. Per un elenco completo, è possibile consultare la voce set nel Capitolo 42.

Il comando set ha una controparte nel comando show, con il quale vengono visualizzate le istruzioni impostate. Per esempio, si possono verificare le impostazioni di feedback digitando:

```
show feedback
```

La risposta visualizzata da SQLPLUS è:

```
FEEDBACK ON for 25 or more rows
```

Anche il parametro dell'ampiezza impostata per visualizzare i numeri è stato modificato con il comando `set`. Si può verificarlo digitando:

```
show numwidth
```

La risposta visualizzata da SQLPLUS è:

```
numwidth 9
```

Dato che il valore 9 rappresenta una larghezza ampia per la visualizzazione di numeri di pagine che non contengono mai più di due cifre, si può restringerla digitando quanto segue:

```
set numwidth 5
```

Ciò tuttavia implica che tutte le colonne di numeri siano di ampiezza pari a cinque caratteri. Se si prevede di dover utilizzare numeri con più di cinque cifre, occorre impostare un valore più alto. Le singole colonne visualizzate possono essere impostate anche indipendentemente l'una dall'altra. Questo argomento verrà trattato nel Capitolo 6.

### 3.4 I comandi `select`, `from`, `where` e `order by`

Sono quattro le parole chiave fondamentali di SQL che vengono utilizzate per selezionare le informazioni da una tabella di Oracle: `select`, `from`, `where` e `order by`. I termini `select` e `from` si utilizzano in tutte le query di Oracle.

La parola chiave `select` indica a Oracle le colonne che si desidera visualizzare, mentre `from` indica i nomi delle tabelle in cui si trovano tali colonne. Riprendendo le procedure illustrate in precedenza con la tabella `GIORNALE`, è possibile osservare come sono stati utilizzati questi comandi. Nella prima linea digitata, ciascun nome di colonna è seguito da una virgola tranne l'ultimo. Si può osservare che la struttura di una query SQL digitata correttamente è decisamente analoga a una frase in lingua naturale. Una query SQLPLUS solitamente termina con il punto e virgola (talvolta definito *terminatore SQL*). La parola chiave `where` indica a Oracle quali qualificatori inserire nelle informazioni selezionate. Per esempio, se si imposta:

```
select Argomento, Sezione, Pagina from GIORNALE
  where Sezione = 'F';
```

ARGOMENTO	S	PAGINA
Nascite	F	7
Annunci	F	8
Necrologi	F	6
Salute	F	6

Viene controllata ogni riga della tabella `GIORNALE` prima della visualizzazione, saltando le righe che non riportano unicamente la lettera 'F' nella colonna `Sezione`. Vengono invece selezionate e visualizzate le righe in cui la `Sezione` è 'F'.

Per indicare a Oracle che si desidera che le informazioni vengano visualizzate in un determinato ordine, si utilizza il comando `order by`. Si possono richiedere vari gradi di selezione nell'ordine. Si considerino questi esempi:

```
select Argomento, Sezione, Pagina from GIORNALE  
where Sezione = 'F';  
order by Argomento;
```

ARGOMENTO	S	PAGINA
Annunci	F	8
Nascite	F	7
Necrologi	F	6
Salute	F	6

Se si ordina per pagina, la sequenza viene quasi capovolta, come illustrato qui di seguito:

```
select Argomento, Sezione, Pagina from GIORNALE  
where Sezione = 'F';  
order by Pagina;
```

ARGOMENTO	S	PAGINA
Salute	F	6
Necrologi	F	6
Nascite	F	7
Annunci	F	8

Nell'esempio seguente, le righe vengono innanzitutto ordinate per Pagina (si osservi nell'elenco precedente la sequenza ordinata soltanto per Pagina). Successivamente vengono ordinate per Argomento, elencando "Necrologi" prima di "Salute".

```
select Argomento, Sezione, Pagina from GIORNALE  
where Sezione = 'F'  
order by Pagina, Argomento;
```

ARGOMENTO	S	PAGINA
Necrologi	F	6
Salute	F	6
Nascite	F	7
Annunci	F	8

Anche con `order by` è possibile invertire l'ordine normale, come in questo caso:

```
select Argomento, Sezione, Pagina from GIORNALE  
where Sezione = 'F'  
order by Pagina desc, Argomento;
```

ARGOMENTO	S	PAGINA
Annunci	F	8
Nascite	F	7
Necrologi	F	6
Salute	F	6

La parola chiave `desc` sta per *descending* (decrescente). In questo caso segue la parola “Pagina” nella riga di `order by`, quindi significa che i numeri delle pagine devono essere visualizzati in ordine decrescente. L’effetto ottenuto sarebbe lo stesso sulla colonna Argomento, se si inserisse la parola chiave `desc` dopo “Argomento” nella riga di `order by`.

Si osservi che i comandi `select`, `from`, `where` e `order by` presentano modalità specifiche nella strutturazione delle parole che seguono. In termini relazionali, i gruppi di parole che comprendono queste parole chiave sono spesso definiti *clausole*, come mostrato nella Figura 3.2.

### 3.5 Logica e valore

Anche la clausola `where`, proprio come `order by`, può essere composta da più parti, ma con un grado di raffinatezza decisamente superiore. Il grado di utilizzo di `where` viene controllato mediante l’uso attento di istruzioni logiche impartite a Oracle in base alle esigenze del momento. Tali istruzioni vengono espresse utilizzando simboli matematici detti *operatori logici*. Questi operatori logici verranno descritti tra breve e sono elencati nel Capitolo 42.

Ecco un semplice esempio di relazione in cui i valori della colonna Pagina vengono controllati per vedere se sono uguali a 6. Viene visualizzata ogni riga per la quale l’uguaglianza risulta vera. Qualsiasi riga in cui Pagina non sia uguale a 6 (in altre parole, le righe in cui l’uguaglianza `Pagina = 6` è falsa) viene saltata.

```
select Argomento, Sezione, Pagina
  from GIORNALE
 where Pagina = 6;
```

ARGOMENTO	S	PAGINA
Necrologi	F	6
Salute	F	6

Il segno di uguale prende il nome di *operatore logico*, perché opera effettuando una verifica logica, ovvero confrontando i valori posti alla sua sinistra e alla sua destra, in questo caso il valore `Pagina` e il valore 6, per controllare se sono uguali.

In questo esempio il valore da verificare non è stato racchiuso tra doppi apici, perché la colonna nella quale il valore dev’essere verificato (la colonna `Pagina`) è stata definita come tipo di dati `NUMBER`. Per i valori numerici non è necessario digitare doppi apici per effettuare la verifica.

### Verifiche di valori singoli

È possibile utilizzare un singolo operatore logico per verificare un singolo valore, come viene spiegato nel riquadro “Test logici su un singolo valore”. Si osservi qualche esempio tra le espres-

```
select Argomento, Sezione, Pagina      <--clausola select
  from GIORNALE                         <--clausola from
 where Sezione = 'F'                     <--clausola where
```

**Figura 3.2** Clausole relazionali.

## Test logici su un singolo valore

**T**utti i seguenti operatori funzionano con lettere o numeri, colonne o letterali.

### **Uguale, maggiore di, minore di, diverso da**

Pagina=	6	Pagina è uguale a 6.
Pagina>	6	Pagina è maggiore di 6.
Pagina>=	6	Pagina è maggiore o uguale a 6.
Pagina<	6	Pagina è minore di 6.
Pagina<=	6	Pagina è minore o uguale a 6.
Pagina!=	6	Pagina è diverso da 6.
Pagina^=	6	Pagina è diverso da 6.
Pagina<>	6	Pagina è diverso da 6.

Dato che su alcune tastiere manca il punto esclamativo o l'accento circonflesso (^), Oracle prevede tre modi per specificare l'operatore di disuguaglianza. L'alternativa finale, <>, va considerata come operatore di disuguaglianza perché filtra i numeri minori di 6 (in questo esempio) o maggiori di 6, ma non il 6.

### **LIKE**

Argomento LIKE 'Vi%'	Argomento inizia con le lettere 'Vi'.
Argomento LIKE '__l%'	Argomento ha una 'l' nella terza posizione.
Argomento LIKE '%0%0%0%'	Argomento contiene due 'o'.

LIKE esegue una ricerca per modelli. Un carattere di sottolineatura rappresenta uno e un solo carattere. Un segno di percentuale rappresenta un numero qualsiasi di caratteri, anche zero.

### **IS NULL, IS NOT NULL**

Precipitazione IS NULL	"Precipitazione è ignoto".
Precipitazione IS NOT NULL	"Precipitazione è noto".

NULL consente di verificare se in una riga di una colonna esistono dati. Se la colonna è completamente vuota, è NULL. Con NULL e NOT NULL va utilizzato IS, perché con essi i segni di uguale, maggiore o minore non funzionano.

sioni elencate nel riquadro: funzionano tutte in maniera analoga e possono essere combinate in relazione alle esigenze, anche se devono essere seguite delle regole specifiche sulle modalità di azione reciproca

### **Uguale, maggiore, minore, diverso**

Con i test logici è possibile confrontare dei valori, sia con il parametro di uguaglianza, sia con parametri relativi. Ecco un semplice test effettuato per tutte le sezioni uguali a B:

```
select Argomento, Sezione, Pagina
  from GIORNALE
 where Sezione = 'B';
ARGOMENTO      S PAGINA
-----
Televisione    B      7
Vita moderna   B      1
Film           B      4
Bridge          B      2
```

Il test seguente riguarda tutti i numeri di pagina maggiori di 4:

```
select Argomento, Sezione, Pagina
  from GIORNALE
 where Pagina > 4;
ARGOMENTO      S PAGINA
-----
Editoriali      A    12
Televisione    B     7
Nascite         F     7
Annunci         F     8
Necrologi       F     6
Salute          F     6
```

Il test seguente riguarda le sezioni maggiori di B (ovvero con lettere che seguono B nell'ordine alfabetico):

```
select Argomento, Sezione, Pagina
  from GIORNALE
 where Sezione > 'B';
```

```
ARGOMENTO      S PAGINA
-----
Sport          D     1
Economia       E     1
Meteo          C     2
Nascite         F     7
Annunci         F     8
Fumetti        C     4
Necrologi       F     6
Salute          F     6
```

Si può impostare un test per valori maggiori di un valore dato, o anche per valori minori, come mostrato di seguito:

```
select Argomento, Sezione, Pagina
  from GIORNALE
 where Pagina < 8;
```

```
ARGOMENTO      S      PAGINA
-----
Notizie         A      1
Sport          D      1
Economia       E      1
Meteo          C      2
Televisione    B      7
Nascite         F      7
Vita moderna   B      1
Fumetti        C      4
Film           B      4
Bridge          B      2
Necrologi       F      6
Salute          F      6
```

L'opposto della verifica di uguaglianza è la verifica “diverso da”, come in questo esempio:

```
select Argomento, Sezione, Pagina
```

```
from GIORNALE
```

```
where Pagina <> 1;
```

ARGOMENTO	S	PAGINA
Editoriali	A	12
Meteo	C	2
Televisione	B	7
Nascite	F	7
Annunci	F	8
Fumetti	C	4
Film	B	4
Bridge	B	2
Necrologi	F	6
Salute	F	6

**NOTA** Occorre prestare particolare attenzione quando si utilizzano gli operatori “maggiori di” e “minore di” per verificare dei numeri che si trovano in colonne impostate con tipo di dati carattere. Tutti i valori nelle colonne di tipo VARCHAR2 e CHAR vengono trattati come caratteri durante le verifiche. Per questo motivo, i numeri che si trovano in questi tipi di colonne vengono considerati come se fossero stringhe di caratteri e non numeri. Se il tipo di dati impostato per una colonna è NUMBER, il valore 12 risulta maggiore del valore 9; se si tratta di una colonna con impostazione non numerica, allora il valore 9 viene considerato maggiore di 12 perché il carattere ‘9’ è più grande del carattere ‘1’.

## LIKE

Una delle caratteristiche logiche più potenti di SQL è un operatore straordinariamente efficace denominato LIKE. Con questo operatore è possibile effettuare una ricerca nelle righe di una colonna di un database per valori che assomigliano a un dato pattern descritto. Vengono utilizzati due caratteri speciali per contrassegnare il tipo di verifica da effettuare: il segno di percentuale, detto *carattere jolly*, e un trattino di sottolineatura detto *marcatore di posizione*. Per trovare tutti gli argomenti che iniziano con la lettera ‘N’ si procede in questo modo:

```
select Argomento, Sezione, Pagina from GIORNALE
```

```
where Argomento LIKE 'N%':
```

ARGOMENTO	S	PAGINA
Nascite	F	7
Necrologi	F	6

Il segno di percentuale (%) significa che qualsiasi carattere può essere accettato in quella posizione: un carattere, cento o nessuno. Se la prima lettera è ‘N’, viene visualizzata la riga trovata mediante l’operatore LIKE. Se invece fosse stata utilizzata ‘n%’ come condizione di ricerca, allora non sarebbe stata visualizzata nessuna riga, poiché le lettere minuscole e maiuscole nei valori dei dati devono essere indicate con precisione. Se si desidera trovare gli argomenti in cui compare la lettera ‘i’ come terza lettera del titolo, senza alcuna condizione riguardo ai due caratteri precedenti e a quelli seguenti, si utilizzano due trattini di sottolineatura ( \_\_ ) per specificare che qualsiasi carattere precedente è accettabile. Nella posizione tre deve trovarsi una ‘i’ minuscola; il segno di percentuale indica che qualsiasi carattere è accettabile nella posizione seguente.

---

```
select Argomento, Sezione, Pagina from GIORNALE
where Argomento LIKE '_i%';
```

ARGOMENTO	S	PAGINA
Editoriali	A	12
Bridge	B	2

Si possono anche utilizzare più segni di percentuale. Per trovare le parole in cui siano presenti due ‘o’ minuscole in qualsiasi posizione, vengono utilizzati tre segni di percentuale, come in questo esempio:

```
select Argomento, Sezione, Pagina from GIORNALE
where Argomento LIKE '%o%%';
```

ARGOMENTO	S	PAGINA
Necrologi	F	6

Si confronti la query con la stessa query, in cui però l’elemento da ricercare è rappresentato da due ‘i’:

```
select Argomento, Sezione, Pagina from GIORNALE
where Argomento LIKE '%i%ii%';
```

ARGOMENTO	S	PAGINA
Editoriali	A	12
Televisione	B	7

Questa funzionalità di ricerca può avere un ruolo importante nel rendere un’applicazione più facile da utilizzare, semplificando le ricerche per nome, prodotto, indirizzo e altri elementi che potrebbero essere ricordati solo parzialmente.

## **NULL e NOT NULL**

Nella tabella GIORNALE non ci sono colonne con il valore NULL, anche se con il comando describe è stato evidenziato che tale valore era consentito. La query che segue relativa alla tabella COMFORT contiene, oltre ad altri dati, le precipitazioni rilevate a San Francisco, in California, e a Keene, nel New Hampshire (Stati Uniti), per quattro date campione durante l’anno 2001.

```
select Citta, DataCampione, Precipitazione
from COMFORT;
```

CITTA	DATACAMPI	PRECIPITAZIONE
SAN FRANCISCO	21-MAR-01	.5
SAN FRANCISCO	22-JUN-01	.1
SAN FRANCISCO	23-SEP-01	.1
SAN FRANCISCO	22-DEC-01	2.3
KEENE	21-MAR-01	4.4
KEENE	22-JUN-01	1.3
KEENE	23-SEP-01	
KEENE	22-DEC-01	3.9

Con la query seguente si possono facilmente scoprire la città e le date in cui le precipitazioni non sono state misurate:

```
select Città, DataCampione, Precipitazione
  from COMFORT;
 where Precipitazione IS NULL;
```

CITTÀ	DATACAMPI	PRECIPITAZIONE
KEENE	23-SEP-01	

IS NULL fondamentalmente indica a Oracle di identificare le colonne in cui il dato è mancante. Non è possibile sapere se in quel giorno il valore era pari a 0, 1, o 5 cm. Poiché il dato è sconosciuto, il valore nella colonna non è indicato come 0, ma la posizione rimane vuota. Utilizzando NOT, si possono anche trovare le città e le date per cui i dati invece esistono, impostando la query seguente:

```
select Città, DataCampione, Precipitazione
  from COMFORT
 where Precipitazione IS NOT NULL;
```

CITTÀ	DATACAMPI	PRECIPITAZIONE
SAN FRANCISCO	21-MAR-01	.5
SAN FRANCISCO	22-JUN-01	.1
SAN FRANCISCO	23-SEP-01	.1
SAN FRANCISCO	22-DEC-01	2.3
KEENE	21-MAR-01	4.4
KEENE	22-JUN-01	1.3
KEENE	22-DEC-01	3.9

È possibile utilizzare gli operatori relazionali (=, != e così via) con NULL, ma questo tipo di condizione non fornisce risultati significativi. Occorre utilizzare IS o IS NOT per operare con NULL.

## Verifiche semplici con un elenco di valori

Se si possono utilizzare operatori logici per verificare un singolo valore, esistono altri operatori logici che si possono utilizzare per verificare diversi valori, come un elenco? Nel riquadro “Test logici su un elenco di valori” viene illustrato proprio questo gruppo di operatori.

Ecco alcuni esempi di come vengono utilizzati questi operatori logici:

```
select Argomento, Sezione, Pagina
  from GIORNALE
 where Sezione IN ('A','B','F');
```

ARGOMENTO	S	PAGINA
Notizie	A	1
Editoriali	A	12
Televisione	B	7
Nascite	F	7
Annunci	F	8
Vita moderna	B	1

---

## Test logici su un elenco di valori

---

**Con numeri:**

Pagina IN (1,2,3)  
 Pagina NOT IN (1,2,3)  
 Pagina BETWEEN 6 AND 10  
 Pagina NOT BETWEEN 6 AND 10

Pagina è nell'elenco (1,2,3).  
 Pagina non è nell'elenco (1,2,3).  
 Pagina è uguale a 6, 10 o qualunque valore tra essi compreso.  
 Pagina è minore di 6 o maggiore di 10.

**Con lettere (o caratteri):**

Sezione IN ('A','C','F')  
 Sezione NOT IN ('A','C','F')  
 Sezione BETWEEN 'B' AND 'D'  
 Sezione NOT BETWEEN 'B' AND 'D'

Sezione è nell'elenco ('A','C','F').  
 Sezione non è nell'elenco ('A','C','F').  
 Sezione è uguale a 'B', 'D' o qualsiasi lettera tra esse compresa (in ordine alfabetico).  
 Sezione è precedente a 'B' o successiva a 'D' (in ordine alfabetico).

Film	B	4
Bridge	B	2
Necrologi	F	6
Salute	F	6

```
select Argomento, Sezione, Pagina
  from GIORNALE
 where Sezione NOT IN ('A','B','F');
```

ARGOMENTO	S	PAGINA
Sport	D	1
Economia	E	1
Meteo	C	2
Fumetti	C	4

```
select Argomento, Sezione, Pagina
  from GIORNALE
 where Pagina BETWEEN 7 and 10;
```

ARGOMENTO	S	PAGINA
Televisione	B	7
Mascite	F	7
Annunci	F	8

Questi test logici possono anche essere combinati, come in questo caso:

```
select Argomento, Sezione, Pagina
  from GIORNALE
 where Sezione = 'F'
   AND Pagina > 7;
```

ARGOMENTO	S	PAGINA
Annunci	F	8

Il comando AND è stato utilizzato per combinare due espressioni logiche; ogni riga esaminata viene verificata per entrambi i parametri; sia “Sezione = ‘F’”, sia “Pagina > 7” devono essere vere perché una riga venga visualizzata. In alternativa si può utilizzare OR: in questo caso vengono visualizzate le righe che soddisfano *almeno una* delle due espressioni logiche:

```
select Argomento, Sezione, Pagina
  from GIORNALE
 where Sezione = 'F'
       OR Pagina > 7;
```

ARGOMENTO	S	PAGINA
-----	-----	-----
Editoriali	A	12
Nascite	F	7
Annunci	F	8
Necrologi	F	6
Salute	F	6

Nell'esempio precedente alcune righe vengono visualizzate anche se la Sezione non è uguale a ‘F’, perché il numero di Pagina è maggiore di 7, mentre altre sono visualizzate anche se il numero di Pagina è minore o uguale a 7, perché la Sezione corrispondente è uguale a ‘F’.

Infine si possono selezionare le righe nella Sezione F tra la pagina 7 e la pagina 10 impostando la seguente query:

```
select Argomento, Sezione, Pagina
  from GIORNALE
 where Sezione = 'F'
       and Pagina BETWEEN 7 AND 10;
```

ARGOMENTO	S	PAGINA
-----	-----	-----
Nascite	F	7
Annunci	F	8

Sono disponibili anche altri *operatori a più valori* il cui utilizzo è più complesso; verranno illustrati nel Capitolo 8. È anche possibile consultare il Capitolo 42 per trovare ulteriori informazioni al riguardo.

## La logica della combinazione

Gli operatori AND e OR si utilizzano seguendo il significato comune dei termini corrispondenti (“e” e “o”). Possono essere combinati in un numero virtualmente infinito di modi, tuttavia occorre prestare attenzione, perché si rischia facilmente di creare query errate.

Si supponga di voler trovare nel giornale le rubriche che risultano più nascoste, quelle che sono da qualche parte dopo la pagina 2 della sezione A o B. Si potrebbe provare così:

```
select Argomento, Sezione, Pagina
  from GIORNALE
 where Sezione = 'A'
       or Sezione = 'B'
       and Pagina > 2;
```

ARGOMENTO	S	PAGINA
Notizie	A	1
Editoriali	A	12
Televisione	B	7
Film	B	4

Si osservi che il risultato ottenuto non è quello desiderato. In qualche modo, la pagina 1 della sezione A è stata inclusa nelle righe visualizzate. Come si spiegano questi risultati? Esiste un modo per ottenere da Oracle una risposta corretta? Anche se gli operatori AND e OR sono entrambi connettori logici, AND è prevalente, perché collega le espressioni logiche alla sua sinistra e alla sua destra in maniera più forte di quanto accada con OR (teoricamente, si dice che ha *precedenza superiore*).

```
where Sezione = 'A'
or Sezione = 'B'
and Pagina > 2;
```

viene interpretata in italiano come “dove Sezione = ‘A’, o dove Sezione = ‘B’ e Pagina > 2”. Se si osserva ciascuno degli esempi con risultato errato proposti in precedenza, si può verificare quale effetto ha avuto questa interpretazione sul risultato. L’operatore AND viene sempre considerato per primo.

Si può superare questo ostacolo utilizzando delle parentesi che racchiudano le espressioni da interpretare insieme. Le parentesi prevalgono sull’ordine di precedenza normale:

```
select Argomento, Sezione, Pagina
  from GIORNALE
 where Pagina > 2 and ( Sezione = 'A'
                        or Sezione = 'B' );
```

ARGOMENTO	S	PAGINA
Editoriali	A	12
Televisione	B	7
Film	B	4

Ora il risultato visualizzato è esattamente quello desiderato. Si osservi che, se si digita la stessa espressione con le sezioni citate all’inizio, il risultato è identico perché le parentesi indicano a Oracle quali sono gli elementi da interpretare insieme. Si confrontino invece i risultati diversi ottenuti cambiando l’ordine nei primi tre esempi precedenti, in cui non venivano utilizzate le parentesi.

### 3.6 Un altro impiego delle subquery con where

Che cosa accadrebbe se gli operatori logici citati nei riquadri “Test logici su un singolo valore” e “Test logici su un elenco di valori” potessero essere utilizzati non soltanto con un unico valore in forma di lettera alfabetica (come ‘F’) o con un elenco di valori digitati al momento (come 4,2,7 o ‘A’, ‘C’, ‘F’), ma anche con valori presi da una query di Oracle? In effetti ciò è possibile utilizzando una caratteristica molto potente di SQL.

Per esempio, si supponga che l'autore dell'articolo di argomento "Salute" pubbli i suoi articoli in diversi giornali, ciascuno dei quali gli invia una copia del sommario in cui è incluso il suo pezzo. Ovviamente l'argomento non viene considerato da tutti i giornali della stessa importanza, viene quindi inserito nella sezione che di volta in volta viene considerata appropriata. Se l'autore non conosce a priori la collocazione del suo argomento, o con quali altre rubriche è stato raggruppato, come potrebbe impostare una query per scoprire la particolare collocazione nel giornale locale? Si potrebbe procedere così:

```
select Sezione from GIORNALE  
where Argomento = 'Salute';
```

```
S  
-  
F
```

Il risultato è 'F'. Conoscendolo, si potrebbe impostare questa query:

```
select ARGOMENTO from GIORNALE  
where Sezione = 'F';
```

```
ARGOMENTO  
-----  
Nascite  
Annunci  
Necrologi  
Salute
```

L'argomento "Salute" è collocato nella sezione F insieme agli altri visualizzati. Le due query impostate potevano essere combinate insieme? Sì, come nell'esempio seguente:

```
select ARGOMENTO from GIORNALE  
where Sezione = (select Sezione from GIORNALE  
                  where Argomento = 'Salute');
```

```
ARGOMENTO  
-----  
Nascite  
Annunci  
Necrologi  
Salute
```

## Valori singoli da una subquery

In effetti, il comando select tra parentesi (definito *subquery*) ha per risultato un valore singolo, F. Nella query principale il valore F viene trattato come se fosse un letterale 'F', come è stato utilizzato nella query precedente. Si ricordi che il segno di uguale rappresenta una verifica di valore singolo e non funziona con elenchi di valori; quindi, se la subquery ha per risultato più di una riga, viene visualizzato un messaggio di errore come questo:

```
select * from GIORNALE  
where Sezione = (select Sezione from GIORNALE  
                  where Pagina = 1);
```

```
ERROR: ORA-1427: single-row subquery returns more than one row
```

Tutti gli operatori logici che si utilizzano per verificare valori singoli possono funzionare nelle subquery, a patto che il risultato di queste sia una riga singola. Per esempio, si può chiedere di visualizzare tutte le rubriche nel giornale che si trovino in sezioni contrassegnate da lettere “minori” (in posizione precedente nell’alfabeto) della sezione in cui si trova l’argomento “Salute”. L’asterisco dopo select è un’abbreviazione per richiedere tutte le colonne in una tabella senza doverle elencare una per una. Le colonne vengono visualizzate nell’ordine in cui sono state create originariamente nella tabella.

```
select * from GIORNALE
where Sezione < (select Sezione from GIORNALE
                    where Argomento = 'Salute');
```

ARGOMENTO	S	PAGINA
Notizie	A	1
Sport	D	1
Editoriali	A	12
Economia	E	1
Meteo	C	2
Televisione	B	7
Vita moderna	B	1
Fumetti	C	4
Film	B	4
Bridge	B	2

10 rows selected.

In questo giornale locale altre dieci rubriche sono poste prima dell’argomento “Salute”.

## Elenco di valori da una subquery

Esattamente come in una subquery possono essere utilizzati gli operatori logici a valore singolo, è anche possibile usare gli operatori a valori multipli. Se una subquery restituisce una o più righe, il valore nella colonna per ciascuna riga viene visualizzato in un elenco. Per esempio, si supponga di voler conoscere le città e i Paesi in cui il tempo è nuvoloso. Si potrebbe avere, come in questo caso, una tabella di informazioni complete sul tempo per tutte le città e una tabella DISLOCAZIONE per tutte le città e i Paesi corrispondenti:

```
select Citta, Paese from DISLOCAZIONE;
```

CITTA	PAESE
ATENE	GRECIA
CHICAGO	STATI UNITI
CONAKRY	GUINEA
LIMA	PERU
MADRAS	INDIA
MANCHESTER	INGHILTERRA
MOSCA	RUSSIA
PARIGI	FRANCIA
SHENYANG	CINA
ROMA	ITALIA
TOKYO	GIAPPONE

SYDNEY	AUSTRALIA
SPARTA	GRECIA
MADRID	SPAGNA

select Citta, Condizione from METEO;

CITTA	CONDIZIONE
LIMA	PIOGGIA
PARIGI	NUVOLOSO
MANCHESTER	NEBBIA
ATENE	SOLE
CHICAGO	PIOGGIA
SYDNEY	NEVE
SPARTA	NUVOLOSO

Innanzitutto si cerca di scoprire in quali città il tempo è nuvoloso:

select Citta from CLIMA  
where Condizione = 'NUVOLOSO';

CITTA
PARIGI
SPARTA

Successivamente si crea un elenco che comprenda quelle città e lo si utilizza per impostare una query riguardante la tabella DISLOCAZIONE:

select Citta, Paese from DISLOCAZIONE  
where Citta IN ('PARIGI', 'SPARTA');

CITTA	PAESE
PARIGI	FRANCIA
SPARTA	GRECIA

Lo stesso risultato si ottiene impostando una subquery, in cui si utilizza il comando select tra parentesi per creare l'elenco di città verificate mediante l'operatore IN, come mostrato di seguito:

select Citta, Paese from DISLOCAZIONE  
where Citta IN (select Citta from CLIMA  
where Condizione = 'NUVOLOSO');

CITTA	PAESE
PARIGI	FRANCIA
SPARTA	GRECIA

Gli altri operatori a valori multipli funzionano in maniera analoga. Il compito fondamentale è quello di creare una subquery che produca un elenco che possa essere verificato logicamente. È importante considerare i seguenti aspetti.

- La subquery deve avere una sola colonna, oppure deve confrontare le colonne selezionate con le diverse colonne poste tra parentesi nella query principale (per ulteriori informazioni, si consulti il Capitolo 12).
- La subquery dev'essere racchiusa tra parentesi.
- Le subquery che hanno come risultato soltanto una riga possono essere utilizzate sia con operatori a valori singoli, sia con operatori a valori multipli.
- Le subquery che hanno come risultato più di una riga possono essere utilizzate soltanto con operatori a valori multipli.

### 3.7 Combinazione di tabelle

Se i dati sono stati normalizzati, probabilmente occorre combinare insieme due o più tabelle per ottenere tutte le informazioni richieste.

Si prenda come esempio l'Oracolo di Delfi. Gli ateniesi chiedono il suo responso sulle forze della natura che possono influire sull'attacco atteso da parte degli spartani e anche sulla direzione dalla quale è probabile che provengano:

```
select Citta, Condizione, Temperatura from CLIMA;
```

CITTA	CONDIZIONE	TEMPERATURA
LIMA	PIOGGIA	7
PARIGI	NUVOLOSO	27
MANCHESTER	NEBBIA	19
ATENE	SOLE	36
CHICAGO	PIOGGIA	19
SYDNEY	NEVE	-2
SPARTA	NUVOLOSO	23

Se sono necessari dei riferimenti geografici più precisi, occorre interrogare la tabella DISLOCAZIONE:

```
select Citta, Longitudine, EstOvest, Latitudine, NordSud
from LOCAZIONE;
```

CITTA	LONGITUDINE E LATITUDINE N
ATENE	23.43 E 37.58 N
CHICAGO	87.38 O 41.53 N
CONAKRY	13.43 O 9.31 N
LIMA	77.03 O 12.03 S
MADRAS	80.17 E 13.05 N
MANCHESTER	2.15 O 53.3 N
MOSCA	37.35 E 55.45 N
PARIGI	2.2 E 48.52 N
SHENYANG	123.3 E 41.48 N
ROMA	12.29 E 41.54 N
TOKYO	139.5 E 35.42 N
SYDNEY	151.1 E 33.52 S
SPARTA	22.27 E 37.05 N
MADRID	3.14 O 40.24 N

Ci sono molte più informazioni di quelle necessarie, però mancano le informazioni sul tempo. Eppure le tabelle CLIMA e DISLOCAZIONE hanno una colonna in comune: Citta. È quindi possibile collegare le informazioni delle due tabelle unendole. È sufficiente utilizzare una query con where per indicare a Oracle ciò che le due tabelle hanno in comune (si tratta di un esempio analogo a quello citato nel Capitolo 1):

```
select CLIMA.Citta, Condizione, Temperatura, Latitudine,
       NordSud, Longitudine, EastOvest
  from CLIMA, LOCAZIONE
 where CLIMA.Citta = LOCAZIONE.Citta;
```

CITTA	CONDIZIONE	TEMPERATURA	LATITUDINE	N	LONGITUDINE	E
ATENE	SOLE	36	37.58	N	23.43	E
CHICAGO	PIOGGIA	19	41.53	N	87.38	0
LIMA	PIOGGIA	7	12.03	S	77.03	0
MANCHESTER	NEBBIA	19	53.3	N	2.15	0
PARIGI	NUVOLOSO	27	48.52	N	2.2	E
SPARTA	NUVOLOSO	23	37.05	N	22.27	E
SYDNEY	NEVE	-2	33.52	S	151.1	E

Si osservi che le righe presenti in questa tabella combinata sono quelle che riportano i valori delle stesse città presenti in entrambe le tabelle originarie. La query con where funziona con la stessa logica esaminata in precedenza nel caso della tabella GIORNALE. Si tratta ovviamente della relazione logica tra le due tabelle che è stata impostata mediante la query digitata. È come dire “seleziona le righe nella tabella CLIMA e nella tabella DISLOCAZIONE in cui le città sono uguali”. Se una città è riportata soltanto in una tabella, manca il termine di uguaglianza nell’altra tabella. La notazione utilizzata con il comando select è TABELLA.NomeColonna, in questo caso CLIMA.Citta.

Mediante il comando select vengono selezionate le colonne delle due tabelle che si desidera vedere visualizzate insieme; qualsiasi colonna in una delle due tabelle che non è stata richiesta viene semplicemente ignorata. Se nella prima riga fosse stato scritto semplicemente:

```
select Citta, Condizione, Temperatura, Latitudine
```

non sarebbe stato possibile per Oracle determinare la città cui si faceva riferimento. In questo caso sarebbe stato visualizzato un messaggio in cui si segnalava il nome di colonna Citta come ambiguo. La sintassi corretta della query con select è “CLIMA.Citta” oppure “DISLOCAZIONE.Citta”. In questo esempio, utilizzare una o l’altra alternativa non fa alcuna differenza, tuttavia vi sono casi in cui colonne con nome identico di due o più tabelle contengono dati molto diversi.

Nella clausola where è inoltre necessario indicare i nomi delle tabelle accanto al nome identico della colonna mediante la quale le due tabelle vengono combinate: “dove tempo punto città equivale a dislocazione punto città”, ovvero, dove la colonna Citta della tabella CLIMA equivale alla colonna Citta nella tabella DISLOCAZIONE.

Si osservi che il prodotto della combinazione delle due tabelle ha l’aspetto di un’unica tabella con sette colonne e sette righe. Tutto ciò che è stato escluso impostando la query non compare nella nuova visualizzazione. Non ci sono colonne Umidità, anche se una colonna con questo nome fa parte della tabella CLIMA. Non ci sono colonne Paese, anche se una colonna con questo nome fa parte della tabella DISLOCAZIONE. Inoltre, delle 14 città della tabella DISLOCAZIONE, sono presenti in questa tabella soltanto quelle della tabella CLIMA. Utilizzando la clausola where, gli altri elementi non sono stati selezionati.

Una tabella creata combinando colonne da più tabelle viene talvolta definita *proiezione*, o *tabella risultante*.

### 3.8 Creazione di una vista

La proiezione o tabella risultante può ricevere un nome ed essere utilizzata proprio come una vera tabella. Questo procedimento viene definito come creazione di una *vista*. Creando una vista viene occultata la logica che stava alla base della tabella combinata. Il tutto funziona in questo modo:

```
create view INVASIONE AS
select CLIMA.Citta, Condizione, Temperatura, Latitudine,
       NordSud, Longitudine, EstOvest
  from CLIMA, DISLOCAZIONE
 where CLIMA.Citta = DISLOCAZIONE.Citta;
```

View created.

Ora si può agire esattamente come se INVASIONE fosse una vera tabella con le proprie righe e colonne. Si può anche impostare una query per ottenere da Oracle la descrizione della nuova tabella:

```
describe INVASIONE
```

Name	Null?	Type
CITTA		VARCHAR2(11)
CONDIZIONE		VARCHAR2(9)
TEMPERATURA		NUMBER
LATITUDINE		NUMBER
NORDSUD		CHAR(1)
LONGITUDINE		NUMBER
ESTOVEST		CHAR(1)

È anche possibile interrogare direttamente la tabella (si osservi che non occorre specificare da quale tabella sono tratte le colonne Citta, perché questa relazione logica è implicita nella vista):

```
select Citta, Condizione, Temperatura, Latitudine, NordSud,
       Longitudine, EstOvest
  from INVASIONE;
```

CITTA	CONDIZIONE	TEMPERATURA	LATITUDINE	N	LONGITUDINE	E
ATENE	SOLE	36	37.58	N	23.43	E
CHICAGO	PIOGGIA	19	41.53	N	87.38	0
LIMA	PIOGGIA	7	12.03	S	77.03	0
MANCHESTER	NEBBIA	19	53.3	N	2.15	0
PARIGI	NUVOLOSO	27	48.52	N	2.2	E
SPARTA	NUVOLOSO	23	37.05	N	22.27	E
SYDNEY	NEVE	-2	33.52	S	151.1	E

Alcune funzioni di Oracle utilizzabili su una tabella comune non possono invece essere usate su una vista, ma sono poche e per lo più riguardano la modifica di righe e l'impostazione di indici di tabelle, argomenti che vengono discussi nei capitoli successivi. Nella maggior parte dei casi, una vista può dare risultati ed essere manipolata esattamente come qualsiasi altra tabella.

**NOTA** *Le viste non contengono dati, che invece sono contenuti nelle tabelle. A partire da Oracle8i è possibile creare “viste materializzate” che contengono dati, e che sono effettivamente delle tabelle, non delle viste.*

Si supponga ora che non sia più necessario avere informazioni su Chicago o altre città fuori dalla Grecia; pertanto, si cerca di modificare in tal senso la query. La seguente impostazione può funzionare?

```
select Citta, Condizione, Temperatura, Latitudine, NordSud,
       Longitudine, EstOvest
  from INVASIONE
 where Paese = 'GRECIA';
```

Viene visualizzato il messaggio seguente:

```
where Paese = 'GRECIA'
*
ERROR at line 4: ORA-00904: invalid column name
```

Questo avviene perché, anche se Paese è una colonna effettiva di una delle tabelle originarie della vista INVASIONE, non si trova nella parte selezionata durante la creazione della vista. È proprio come se non esistesse. Quindi, occorre tornare al comando `create view` per includere nella nuova tabella soltanto la Grecia.

```
create or replace view INVASIONE as
select METEO.Citta, Condizione, Temperatura, Latitudine,
       NordSud, Longitudine, EstOvest
  from METEO, DISLOCAZIONE
 where METEO.Citta = DISLOCAZIONE.Citta
   and Paese = 'GRECIA';
```

View created.

Utilizzando il comando `create or replace view` è possibile creare una nuova versione della vista senza perdere quella precedente. Con questo comando risulta più semplice amministrare i privilegi di utente per accedere alla vista, come viene descritto nel Capitolo 19.

La logica della clausola `where` ora è stata ampliata per includere entrambe le tabelle di origine e un test a valore singolo su una colonna in una delle tabelle origine. Ora è possibile impostare la query a Oracle e ottenere questa risposta:

```
select Citta, Condizione, Temperatura, Latitudine, NordSud,
       Longitudine, EstOvest
  from INVASIONE;
```

CITTA	CONDIZIONE	TEMPERATURA	LATITUDINE	N	LONGITUDINE	E
ATENE	SOLE	36	37.58	N	23.43	E
SPARTA	NUVOLOSO	23	37.05	N	22.27	E

In questo modo è possibile avvertire gli ateniesi della probabile apparizione degli spartani da sud-ovest, accaldati e stanchi per la marcia. Impostando qualche operazione di trigonometria è anche possibile calcolare la distanza percorsa. L'antico Oracolo di Delfi era sempre ambiguo nelle sue previsioni; in questo caso avrebbe detto: "Gli spartani gli ateniesi conquisteranno." Con Oracle è possibile offrire almeno qualche dato di fatto.

## Espansione di una vista

La caratteristica delle viste di poter occultare o anche modificare i dati può essere impiegata per numerosi scopi utili. Report molto complessi possono essere creati impostando una serie di viste semplici ed elementi o gruppi specifici possono essere delimitati in modo da visualizzare soltanto determinate parti dell'intera tabella.

In effetti, qualunque qualificazione che sia possibile inserire in una query può diventare parte di una vista. Si potrebbe per esempio fare in modo che i supervisori di una tabella degli stipendi vedano soltanto il proprio salario e quello delle persone che lavorano per loro, oppure restringere la visione per i settori operativi di una società in modo che venga visualizzato soltanto il loro risultato finanziario, anche se la tabella in realtà contiene i risultati di tutti i settori. Cosa ancora più importante, le viste *non* sono degli snapshot dei dati fissati in un determinato punto nel passato, ma sono elementi dinamici e riflettono costantemente i dati delle tabelle di origine. Nel momento in cui viene variato un dato in una tabella, qualsiasi vista creata con quella tabella viene conseguentemente modificata.

Per esempio, è possibile creare una vista che limiti i valori sulla base dei valori di una colonna. Come mostrato di seguito, una query che limiti la tabella DISLOCAZIONE alla colonna Paese potrebbe essere utilizzata per limitare le righe visibili nella vista:

```
create or replace view DISLOCAZIONI_PERU as
select * from DISLOCAZIONE
where Paese = 'PERU';
```

Se l'utente imposta la query per DISLOCAZIONI\_PERU, non gli sarà possibile visualizzare nessuna riga di un paese diverso dal Perù.

Le query utilizzate per definire le viste possono anche fare riferimento a *pseudocolonne*. Una pseudocolonna è una "colonna" che restituisce un valore quando viene selezionata, ma non è un'autentica colonna di una tabella. Selezionando la pseudocolonna User, viene sempre visualizzato il nome dell'utente di Oracle che ha impostato la query. Pertanto, se una colonna della tabella contiene nomi utente, tali valori potranno essere confrontati con la pseudocolonna User per applicare restrizioni alle relative righe, come nell'esempio seguente. In tale esempio la query viene eseguita sulla tabella NOME. Se il valore della colonna omonima è identico al nome dell'utente che ha impostato la query, allora vengono visualizzate le righe.

```
create or replace view NOMI_RISTRETTI
select * from NOME
where Nome = User;
```

Questo tipo di vista risulta molto utile quando gli utenti richiedono l'accesso a determinate righe di una tabella; in questo modo non possono vedere tutte le righe che corrispondono al loro nome utente per Oracle.

Le viste sono strumenti molto potenti. Questo argomento verrà trattato in maniera più approfondita nel Capitolo 18.

La clausola `where` può essere utilizzata per unire due tabella che hanno una colonna comune. L'insieme di dati risultante può essere trasformato in una vista (con una nuova denominazione), che può essere trattata come se fosse a sua volta una tabella normale. L'efficacia di una vista si rivela nella possibilità di limitare o modificare le modalità di visualizzazione dei dati da parte di un utente, mantenendo immutate le tabelle originarie.

## Capitolo 4

# Elementi di base dei database relazionali a oggetti

- 4.1 **È obbligatorio utilizzare gli oggetti?**
- 4.2 **Perché utilizzare gli oggetti?**
- 4.3 **Tutti possiedono degli oggetti**
- 4.4 **Un esempio di oggetto comune**
- 4.5 **Analisi e progettazione  
orientata agli oggetti**
- 4.6 **I prossimi capitoli**

**E** possibile ampliare il proprio database relazionale in modo che comprenda concetti e strutture orientate agli oggetti. In questo capitolo viene presentata una panoramica delle principali caratteristiche di questo tipo (introdotte per la prima volta in Oracle8), mostrando l'impatto che hanno sul linguaggio SQL. Le caratteristiche orientate agli oggetti di tipo più avanzato saranno descritte in maniera dettagliata nei capitoli successivi; questo capitolo si limita a introdurre i concetti e a fornire una panoramica generale.

### 4.1 È obbligatorio utilizzare gli oggetti?

Il fatto di utilizzare Oracle non obbliga certo a impiegare concetti di programmazione a oggetti (OOP, Object Oriented Programming) nell'implementazione del proprio database. In effetti, il database è di tipo ORDBMS, un sistema di gestione di database relazionale a oggetti. Ciò implica per gli sviluppatori la disponibilità di tre diverse versioni di Oracle.

Relazionale	Il sistema di gestione di database relazionale di Oracle (RDBMS) descritto nei capitoli precedenti.
Relazionale a oggetti	Il database relazionale di Oracle, ampliato in modo da comprendere concetti e strutture orientate agli oggetti come tipi di dati astratti, tabelle annidate e array variabili.
Orientato agli oggetti	Database orientato agli oggetti il cui sviluppo è basato esclusivamente su analisi e progettazione a oggetti.

Oracle offre supporto completo per tutte e tre le diverse implementazioni. Se si ha già familiarità con Oracle come database relazionale, si può continuare a utilizzarlo nella stessa maniera. Dal momento che le funzionalità di programmazione a oggetti sono ampliamenti rispetto a un database relazionale, è possibile selezionare quali caratteristiche di questo tipo si desidera utilizzare quando si potenziano le applicazioni relazionali esistenti. Se si desidera sviluppare di nuovo e implementare la propria applicazione utilizzando soltanto caratteristiche a oggetti, è possibile scegliere anche questa opzione. A prescindere dal metodo prescelto, occorre innanzi tutto acqui-

sire familiarità con le funzioni e le caratteristiche specifiche di Oracle come database relazionale. Anche se si è pianificato di utilizzare soltanto le capacità di programmazione a oggetti, occorre comunque conoscere le funzioni e i tipi di dati disponibili in Oracle, così come i linguaggi di programmazione utilizzati (SQL e PL/SQL).

In questo capitolo vengono illustrate le parti fondamentali del discorso per SQL con l'aggiunta delle strutture relazionali a oggetti. Nei capitoli seguenti le funzioni di Oracle vengono descritte in maniera più dettagliata, con paragrafi dedicati a PL/SQL (il linguaggio di programmazione procedurale di Oracle), trigger e procedure. Dopo i capitoli dedicati a PL/SQL e alle procedure si troveranno diversi capitoli che affrontano l'implementazione delle caratteristiche di programmazione a oggetti. È importante comprendere le funzioni, le strutture e i linguaggi di programmazione di Oracle prima di implementare le strutture OOP e relazionali a oggetti più avanzate.

## 4.2 Perché utilizzare gli oggetti?

Dato che non è indispensabile utilizzare gli oggetti, perché mai lo si dovrebbe fare? Inizialmente può sembrare che l'utilizzo di caratteristiche a oggetti complichi la struttura e l'implementazione dei propri sistemi di database, proprio come il fatto di aggiungere nuove caratteristiche a qualsiasi sistema può automaticamente aumentarne la complessità. I fautori di queste caratteristiche OOP sostengono che gli oggetti riducono la complessità consentendo all'utente un approccio intuitivo nella rappresentazione di dati complessi e delle relazioni fra di essi. Per esempio, se si desidera spostare un'automobile (un oggetto), la si può trasferire direttamente, oppure si possono prendere le parti che la compongono (pneumatici, piantone dello sterzo e così via), spostarle singolarmente e quindi ricomporle nella nuova collocazione. Trattare l'automobile come un oggetto è una modalità di approccio più naturale che semplifica l'interazione.

Oltre a semplificare le interazioni con i dati, gli oggetti risultano utili anche in altri modi. In questo capitolo vengono illustrati alcuni esempi di tre vantaggi che derivano dall'utilizzo di alcune caratteristiche orientate agli oggetti.

- **Riutilizzo degli oggetti.** Se si scrive codice a oggetti, si incrementano le possibilità di riutilizzare moduli scritti in precedenza. Analogamente, se si creano oggetti di database di tipo orientato agli oggetti, aumentano le possibilità che questi possano essere riutilizzati.
- **Aderenza allo standard.** Se si creano oggetti standard, aumenta la possibilità che possano essere riutilizzati. Se in applicazioni o tabelle multiple vengono utilizzati gli stessi gruppi di oggetti di database, viene creato uno standard de facto. Per esempio, se si creano tipi di dati standard da utilizzare per tutti gli indirizzi, per tutti gli indirizzi nel database verrà utilizzato lo stesso formato interno.
- **Percorsi di accesso definiti.** Per ciascun oggetto è possibile definire le procedure e le funzioni che possano influire su di esso, quindi si possono unire i dati e i metodi per accedervi. Definendo i percorsi di accesso in questo modo, è possibile standardizzare i metodi di accesso ai dati e incrementare la possibilità di riutilizzare gli oggetti.

Gli svantaggi dell'utilizzo di oggetti sono rappresentati principalmente dalla complessità aggiunta al sistema e dal tempo necessario per imparare a implementarne le caratteristiche. Tuttavia, come si osserverà in questo capitolo, i principi fondamentali per modificare l'RDBMS Oracle in modo da includere le funzionalità a oggetti si basano semplicemente sul modello relazionale presentato nei primi capitoli. Il breve tempo impiegato per sviluppare e utilizzare tipi di dati astratti, come si vedrà più avanti nel capitolo, dovrebbe consentire di bilanciare il tempo impiegato per imparare le caratteristiche orientate agli oggetti.

## 4.3 Tutti possiedono degli oggetti

Tutti possiedono dei dati e dei metodi per interagire con essi. Dal momento che un oggetto è costituito dalla combinazione di dati e metodi, tutti possiedono degli oggetti. Si consideri l'esempio della biblioteca presentato nei capitoli precedenti. Se si tiene traccia di chi prende in prestito i libri, ogni voce conterrà informazioni di base sugli utenti, come nome, indirizzo ed eventualmente tipo preferito di libro. In genere vi sarà uno standard per la struttura degli indirizzi: prima il nome della persona, poi il nome della strada, il nome della città e quello della provincia. Quando viene aggiunta una nuova persona alla lista degli utenti che prendono libri in prestito, viene sempre seguita la stessa procedura. Ecco alcuni dei metodi possibili per aggiungere un nuovo utente all'elenco.

Aggiungi_Utente	Per aggiungere una persona all'elenco.
Aggiorna_Utente	Per aggiornare i dati relativi a una persona.
Elimina_Utente	Per cancellare una persona dall'elenco.
Conta_Utenti	Per contare il numero di utenti per tipo preferito di libro.

Come mostra il metodo Conta\_Utenti, i metodi non devono essere utilizzati per manipolare i dati, ma per creare report sulla base di questi. Si possono inoltre impostare delle funzioni con i dati e fare in modo che il risultato della funzione arrivi all'utente. Per esempio, se si fossero memorizzate le date di nascita, si potrebbe utilizzare un metodo Età per calcolare e creare report su tali dati.

Oracle supporta molti tipi diversi di oggetti, descritti nei paragrafi seguenti.

### Tipi di dati astratti

I *tipi di dati astratti* sono tipi di dati costituiti da uno o più sottotipi. Anziché limitarsi ai tipi di dati standard di Oracle, NUMBER, DATE e VARCHAR2, i tipi di dati astratti possono consentire di descrivere in modo più accurato i propri dati. Per esempio, un tipo di dato astratto per contenere indirizzi può essere rappresentato dalle colonne seguenti:

Via	VARCHAR2(50)
Citta	VARCHAR2(25)
Prov	CHAR(2)
CAP	NUMBER

Quando si crea una tabella che contiene anche informazioni relative all'indirizzo, è possibile utilizzare una colonna con impostato un tipo di dati astratto per gli indirizzi che contenga a sua volta le colonne Via, Citta, Prov e Cap quali sue parti. I tipi di dati astratti possono essere annidati e possono contenere riferimenti ad altri tipi di dati astratti. Nel paragrafo seguente viene presentato un esempio dettagliato di tipi di dati astratti annidati.

Due dei vantaggi elencati in precedenza relativamente all'utilizzo di oggetti, il riutilizzo e l'aderenza allo standard, vengono realizzati utilizzando tipi di dati astratti. Quando viene creato un tipo di dati di questo genere, si crea uno standard per la rappresentazione di elementi di dati astratti (come indirizzi, persone o società). Se si utilizza lo stesso tipo di dati astratto in più posizioni, si può essere certi che gli stessi dati logici vengano rappresentati nella stessa maniera in quelle posizioni.

È possibile utilizzare tipi di dati astratti per creare *tabelle di oggetti*. In una tabella di oggetti, le colonne corrispondono a quelle del tipo di dati astratto.

## Tabelle annidate

Una *tabella annidata* è formata da un’insieme di righe, rappresentate come una colonna all’interno della tabella principale. Per ciascuna voce all’interno della tabella principale, la tabella annidata può contenere più righe. In un certo senso si tratta di un modo per memorizzare più relazioni all’interno di una tabella. Si consideri per esempio una tabella che contenga informazioni sui settori di una azienda, ciascuno dei quali può avere contemporaneamente diversi progetti in corso. In un modello relazionale rigido occorrerebbe impostare due tabelle diverse, SETTORE e PROGETTO.

Utilizzando una tabella annidata è possibile memorizzare le informazioni relative ai progetti all’interno della tabella SETTORE. Si può accedere alle voci della tabella PROGETTO direttamente tramite la tabella SETTORE, senza necessità di effettuare un collegamento. La possibilità di selezionare i dati senza doversi spostare tra collegamenti rende più facile per l’utente l’accesso ai dati stessi. Anche se non vengono definiti i metodi per accedere ai dati annidati, i dati relativi ai settori e ai progetti sono stati chiaramente associati. In un modello relazionale rigido, l’associazione tra le tabelle SETTORE e PROGETTO sarebbe stata effettuata impostando relazioni tra chiavi esterne. Esempi di tabelle annidate e altri tipi di collection sono riportati nella Parte quarta.

## Array variabili

Un *array variabile* è, come una tabella annidata, una collection; si tratta di un insieme di oggetti dello stesso tipo di dati, visto come colonna all’interno di una tabella. Quando viene creato un array, vengono definite anche le sue dimensioni. Quando viene creato un array variabile in una tabella, tale array viene trattato come se fosse una colonna. Concettualmente, un array variabile è una tabella annidata con un numero di righe limitato.

Gli array variabili, detti anche VARRAY, consentono di memorizzare nelle tabelle valori ripetuti. Per esempio, si supponga di avere una tabella PROGETTO e un gruppo di lavoratori assegnati ai vari progetti. In questo sistema, un progetto può essere collegato a molti lavoratori e un lavoratore può lavorare su più progetti. In un’implementazione rigidamente relazionale, è possibile creare una tabella PROGETTO, una tabella LAVORATORE e una tabella di intersezione PROGETTO\_LAVORATORE in cui siano visualizzate le relazioni tra di esse.

È invece possibile utilizzare array variabili per memorizzare i nomi dei lavoratori nella tabella PROGETTO. Se i progetti sono limitati a dieci lavoratori o meno, è possibile creare un array variabile con un limite di dieci voci inserite. Si può utilizzare per l’array variabile qualsiasi tipo di dati appropriato per i nomi dei lavoratori. L’array variabile può essere riempito in modo che per ciascun progetto sia possibile selezionare i nomi di tutti i lavoratori correlati, senza dover interrogare la tabella LAVORATORE. A ogni record di progetto nella tabella PROGETTO corrisponderanno più voci nell’array variabile che contiene i nomi dei lavoratori. Esempi di array variabili e altri tipi di collection sono riportati nella Parte quarta.

## Tipo di dati LOB (Large Object)

Un tipo di dati *large object*, o LOB, è in grado di contenere enormi volumi di dati. I tipi di dati LOB disponibili sono BLOB, CLOB, NCLOB e BFILE. Il tipo di dati BLOB viene utilizzato per dati binari e può arrivare a una lunghezza di 4 GB. Con il tipo di dati CLOB è anche possibile memorizzare dati formati da caratteri fino a un massimo di 4 GB. Il tipo di dati NCLOB viene utilizzato per memorizzare dati CLOB per set di caratteri a più byte. I dati per i tipi BLOB,

CLOB e NCLOB vengono memorizzati all'interno del database, perciò è possibile avere nel database un'unica riga lunga più di 4 GB.

Il quarto tipo di dati LOB, BFILE, funziona come indicatore rispetto a un file esterno. I file di riferimento di BFILE sono situati nel sistema operativo, mentre il database contiene soltanto un indicatore per essi. Le dimensioni dei file esterni sono limitate soltanto dal sistema operativo. Dal momento che i dati sono conservati all'esterno del database, non è possibile mantenerne la corrispondenza o l'integrità.

È possibile impostare più dati LOB per ciascuna tabella. Per esempio, si potrebbe avere una tabella con una colonna CLOB e due colonne BLOB. Si tratta di un perfezionamento del tipo di dati LONG, poiché è possibile avere un'unica colonna LONG per ogni tabella. Con Oracle sono disponibili numerose funzioni e procedure per manipolare e selezionare i dati LOB. Nella Parte quarta è possibile trovare i dettagli riguardanti l'implementazione dei tipi LOB.

## Riferimenti

Le tabelle annidate e gli array variabili sono *oggetti incorporati*. Altri tipi di oggetti, gli *oggetti referenziati*, sono fisicamente separati dagli oggetti che vi fanno riferimento. I riferimenti, o REF, sono sostanzialmente indicatori di oggetti riga. Un "oggetto riga" è diverso da un "oggetto colonna". Un esempio di "oggetto colonna" è un array variabile, un oggetto che viene trattato come colonna all'interno di una tabella. Un "oggetto riga", invece, rappresenta sempre una riga.

L'implementazione dei riferimenti viene descritta nella Parte quarta. Come si è osservato in precedenza, non è necessario utilizzare tutte le funzioni orientate agli oggetti disponibili in Oracle. I riferimenti normalmente sono tra le ultime caratteristiche orientate agli oggetti implementate quando si passa da un'applicazione relazionale a un'applicazione relazionale a oggetti o ad un approccio a oggetti.

## Viste oggetti

Le *viste oggetti* consentono di aggiungere strutture a oggetti a tabelle relazionali esistenti. Per esempio, è possibile creare un tipo di dati astratto sulla base di una definizione di tabella già esistente. Le viste oggetti presentano i vantaggi della strutturazione di una tabella relazionale e delle strutture orientate agli oggetti. Inoltre questa modalità consente di iniziare a sviluppare caratteristiche a oggetti nell'ambito del proprio database relazionale, una sorta di ponte tra il mondo relazionale e quello orientato agli oggetti.

Prima di utilizzare queste caratteristiche, è preferibile acquisire familiarità con i tipi di dati astratti e i trigger. Per una descrizione completa delle viste oggetti, si rimanda alla Parte quarta.

## Convenzioni di denominazione per gli oggetti

Il Capitolo 2 ha trattato l'argomento degli standard di denominazione per tabelle e colonne. In generale occorre utilizzare termini comuni e descrittivi per nomi di tabella e di colonna. Negli esempi di procedure a oggetti citati in questo libro vengono applicate le regole seguenti.

- I nomi di tabella e di colonna sono singolari (come nel caso di IMPIEGATO, Nome e Prov).
- I tipi di dati astratti devono essere sostantivi singolari con suffisso \_TY (come per PERSONA\_TY o INDIRIZZO\_TY).
- I nomi di tabella e di tipi di dati sono sempre in caratteri maiuscoli (come IMPIEGATO o PERSONA\_TY).

- I nomi di colonna iniziano sempre con lettera maiuscola (come Prov e \_Data).
- Le viste oggetti sono sostantivi singolari con suffisso \_OV (come PERSONA\_OV o INDIRIZZO\_OV).
- I nomi di tabelle annidate sono sostantivi plurali con suffisso \_NT (come LAVORATORI\_NT).
- I nomi di array variabili sono sostantivi plurali con suffisso \_VA (come LAVORATORI\_VA).

La denominazione di un oggetto dovrebbe essere costituita da due parti: il nome vero e proprio e il suffisso. Il nome vero e proprio dell'oggetto dovrebbe seguire gli standard di denominazione dell'utente e il suffisso dovrebbe essere d'ausilio per identificare tipi di oggetti particolari.

#### 4.4 Un esempio di oggetto comune

Si consideri un oggetto che si trova nella gran parte dei sistemi: l'indirizzo. Anche in un sistema semplice come quello della biblioteca gli indirizzi vengono conservati e selezionati. Questi indirizzi seguono un formato standard: via, città e provincia. Questo formato standard può essere utilizzato come base per un tipo di dati astratto per gli indirizzi. In primo luogo occorre espanderlo fino a comprendere le informazioni aggiuntive relative all'indirizzo, come il codice postale. Quindi occorre utilizzare il comando `create type` per creare un tipo di dati astratto:

```
create type INDIRIZZO_TY as object
(Via      VARCHAR2(50),
Città    VARCHAR2(25),
Prov     CHAR(2),
Cap      NUMBER);
/
```

Questo comando è il più importante nei database relazionali a oggetti. In questo esempio, con il comando `create type` viene data la seguente indicazione: "crea un tipo di dati astratto di nome INDIRIZZO\_TY, con quattro attributi denominati Via, Città, Prov e Cap, utilizzando per ciascuna colonna i tipi di dati e le lunghezze definite". Finora nessun metodo è stato creato dall'utente, viceversa sono stati creati internamente dal database i metodi utilizzati ogni volta in cui si accede all'oggetto INDIRIZZO\_TY. Per ulteriori informazioni sulla creazione di metodi da parte dell'utente, si può consultare la Parte quarta e la voce `create type body` nel Capitolo 42.

All'interno del comando `create type` la clausola `as object` identifica esplicitamente INDIRIZZO\_TY come un'implementazione di tipo a oggetti.

Ora che è stato creato il tipo di dati INDIRIZZO\_TY, è possibile utilizzarlo nell'ambito di altri tipi di dati. Per esempio, si potrebbe decidere di creare un tipo di dati standard per le persone. Le persone hanno nomi e indirizzi, pertanto si potrebbe creare il tipo seguente:

```
create type PERSONA_TY as object
(Nome      VARCHAR2(25),
Indirizzo INDIRIZZO_TY);
/
```

Che cosa accade con questo comando? Innanzitutto viene attribuito un nome al tipo di dati, PERSONA\_TY, che viene identificato come oggetto con la clausola `as object`. Quindi vengono definite due colonne. In questa linea:

(Nome VARCHAR2(25),

viene definita la prima colonna della rappresentazione di PERSONA\_TY. La linea successiva:

```
Indirizzo  INDIRIZZO_TY);
```

definisce la seconda colonna. Per la seconda colonna, Indirizzo, viene utilizzato il tipo di dati astratto INDIRIZZO\_TY creato in precedenza. Quali sono le colonne all'interno di INDIRIZZO\_TY? Secondo la definizione di INDIRIZZO\_TY, sono le seguenti:

```
create type INDIRIZZO_TY as object
(Via      VARCHAR2(50),
Citta    VARCHAR2(25),
Prov     CHAR(2),
Cap      NUMBER);
/
```

In base a queste definizioni, una voce di PERSONA\_TY è composta dalle colonne Nome, Via, Citta, Prov e Cap, anche se soltanto una di queste colonne è stata esplicitamente impostata con la definizione del tipo di dati PERSONA\_TY.

Si può immaginare come questa opportunità di definire e riutilizzare tipi di dati astratti possa semplificare la rappresentazione dei dati all'interno del proprio database. Per esempio, una colonna denominata Via raramente viene utilizzata in modo autonomo; è quasi sempre utilizzata come parte di un indirizzo. Con un tipo di dati astratto è possibile unire gli elementi e operare con l'intera entità (l'indirizzo) invece che con le singole parti (la colonna Via e le altre che formano l'indirizzo).

Ora è possibile utilizzare il tipo di dati PERSONA\_TY per la creazione della tabella.

## **La struttura di un oggetto semplice**

Per quanto si possa tentare, non è possibile inserire dei dati in PERSONA\_TY. Il motivo è semplice: un tipo di dati descrive i dati, non serve per memorizzarli. Non è possibile memorizzare dei dati in un tipo di dati NUMBER e non è neppure possibile effettuare la stessa operazione con qualunque tipo di dati creato. Per memorizzare dei dati, occorre creare una tabella che utilizzi il tipo di dati desiderato.

Con il comando seguente viene creata una tabella denominata CLIENTE. Ogni cliente ha un identificativo (Cliente\_ID) e tutti gli altri attributi normalmente impostati per le persone (utilizzando il tipo di dati PERSONA\_TY).

```
create table CLIENTE
(Cliente_ID NUMBER,
Persona    PERSONA_TY);
```

Che cosa accade se si utilizza il comando describe per la tabella CLIENTE? Vengono visualizzate le seguenti definizioni di colonna:

```
describe CLIENTE
```

Name	Null?	Type
CLIENTE_ID		NUMBER
PERSONA		PERSONA_TY

Il comando `describe` evidenzia che la colonna Persona della tabella CLIENTE è stata definita utilizzando il tipo di dati PERSONA\_TY. È possibile consultare ulteriormente il dizionario di dati per visualizzare la struttura del tipo di dati PERSONA\_TY. Le colonne di un tipo di dati astratto vengono definite come *attributi* del tipo di dati stesso. Nell'ambito del dizionario di dati, la vista USER\_TYPE\_ATTRS consente di visualizzare informazioni relative agli attributi dei tipi di dati astratti di un utente.

**NOTA** *Il dizionario di dati è costituito da una serie di tabelle e viste che contengono informazioni sulle strutture e gli utenti del database. È possibile impostare delle query per ottenere informazioni utili sugli oggetti del database propri e su quelli a cui è possibile accedere. Si consulti la Parte quarta per una guida “orientata all’utente” relativa alle viste del dizionario dati disponibili.*

Nella query seguente, il nome, la lunghezza e il tipo di dati sono selezionati per ciascuno degli attributi relativi al tipo di dati PERSONA\_TY:

```
select Attr_Name,
       Length,
       Attr_Type_Name
  from USER_TYPE_ATTRS
 where Type_Name = 'PERSONA_TY';

ATTR_NAME          LENGTH ATTR_TYPE_NAME
-----           -----
NOME                  25  VARCHAR2
INDIRIZZO          INDIRIZZO_TY
```

Nel risultato della query viene indicato che il tipo di dati PERSONA\_TY è formato da una colonna Nome (definita come VARCHAR2 con lunghezza 25) e una colonna Indirizzo (definita utilizzando il tipo di dati INDIRIZZO\_TY). Si possono impostare nuove query per USER\_TYPE\_ATTRS in modo da visualizzare gli attributi del tipo di dati INDIRIZZO\_TY:

```
select Attr_Name,
       Length,
       Attr_Type_Name
  from USER_TYPE_ATTRS
 where Type_Name = 'INDIRIZZO_TY';

ATTR_NAME          LENGTH ATTR_TYPE_NAME
-----           -----
VIA                   50  VARCHAR2
CITTA                 25  VARCHAR2
PROV                   2   CHAR
CAP                    1   NUMBER
```

È anche possibile descrivere i tipi di dati astratti con il comando `describe`:

```
desc PERSONA_TY

Name          Null? Type
-----        -----
NOME          VARCHAR2(25)
INDIRIZZO      INDIRIZZO_TY
```

```
desc INDIRIZZO_TY
```

Name	Null?	Type
VIA		VARCHAR2(50)
CITTA		VARCHAR2(25)
PROV		CHAR(2)
CAP		NUMBER

Non accade spesso di dover completamente scomporre i tipi di dati che costituiscono una tabella; tuttavia, se è necessario farlo, è possibile utilizzare il comando `describe` o le query presentate in questo paragrafo per scendere nei livelli di astrazione. Una volta note le strutture di ciascuno dei tipi di dati astratti utilizzati per la tabella, è possibile inserire i record nella tabella stessa.

**NOTA** *Se l'utente non è il proprietario delle tabelle e dei tipi di dati in relazione ai quali vengono cercate le informazioni, è possibile impostare delle query utilizzando i parametri `ALL_TAB_COLUMNS` e `ALL_TYPE_ATTRS` al posto di `USER_TAB_COLUMNS` e `USER_TYPE_ATTRS`. Le viste `ALL_TAB_COLUMNS` e `ALL_TYPE_ATTRS` consentono di visualizzare tutte le colonne e gli attributi delle tabelle e i tipi di dati che sono di proprietà dell'utente o a cui l'utente può accedere. `ALL_TAB_COLUMNS` e `ALL_TYPE_ATTRS` contengono entrambe una colonna denominata `Owner` in cui sono identificati i proprietari delle tabelle o dei tipi di dati. Per una descrizione delle viste del dizionario dati, si rimanda alla Parte quarta.*

**NOTA** *È possibile utilizzare il comando `set describe depth` per vedere i tipi dipendenti e gli attributi quando si descrive una tabella. Vedere il Capitolo 30.*

## Inserimento di record nella tabella CLIENTE

Quando viene creato un tipo di dati astratto, Oracle crea dei metodi per la gestione dei dati detti *costruttori*. Si tratta di programmi che vengono denominati secondo il tipo di dati; i parametri sono costituiti dai nomi degli attributi definiti per il tipo di dati in questione. Quando si devono inserire dei record in una tabella basata su tipi di dati astratti, si impiegano tali metodi. Per esempio, la tabella CLIENTE utilizza il tipo di dati PERSONA\_TY, per il quale viene utilizzato il tipo di dati INDIRIZZO\_TY. Per inserire un record nella tabella CLIENTE è necessario inserire un record ricorrendo al costruttore dei tipi di dati PERSONA\_TY e INDIRIZZO\_TY.

Nell'esempio seguente viene inserito un record nella tabella CLIENTE utilizzando i metodi costruttori per i tipi di dati PERSONA\_TY e INDIRIZZO\_TY. Tali metodi sono evidenziati in grassetto nell'esempio e hanno la stessa denominazione dei tipi di dati:

```
insert into CLIENTE values
(1,
PERSONA_TY('NEIL MULLANE',
INDIRIZZO_TY('57 MT PLEASANT ST',
'FINN', 'NH', 11111)));
```

1 row created.

Con il comando `insert` vengono forniti i valori da inserire in una riga della tabella CLIENTE. I valori indicati devono corrispondere ai tipi di dati delle colonne della tabella.

In questo esempio viene specificato un valore di Cliente\_ID pari a 1. Successivamente vengono impostati i valori per la colonna Persona, utilizzando il costruttore di PERSONA\_TY (in grassetto). Con il tipo di dati PERSONA\_TY viene specificato un Nome, quindi viene utilizzato il metodo INDIRIZZO\_TY per inserire i valori relativi all'Indirizzo. Quindi, per il record inserito nell'esempio il valore di Nome è 'NEIL MULLANE' e il valore di Via è '57 MT PLEASANT ST'. Si osservi che i parametri per il metodo costruttore sono riportati nello stesso ordine degli attributi del tipo di dati.

A questo punto può essere inserito un secondo record nella tabella CLIENTE, utilizzando esattamente la medesima impostazione per richiamare i costruttori (anche in questo caso evidenziati in grassetto):

```
insert into CLIENTE values
(2,
PERSONA_TY('SEYMOUR HESTER',
INDIRIZZO_TY('1 STEPAHEAD RD',
'BRIANT', 'NH', 11111)));
```

Ora è stato inserito il secondo record nella tabella CLIENTE. È sufficiente utilizzare i costruttori quando si manipolano i record nelle tabelle in cui sono stati impostati tipi di dati astratti.

## Selezione da tipi di dati astratti

Se si desidera selezionare i valori di Cliente\_ID per la tabella CLIENTE, è sufficiente impostare una query per i valori di quella colonna della tabella:

```
select Cliente_ID
  from CLIENTE;
CLIENTE_ID
-----
1
2
```

Impostare questo tipo di query per i valori di Cliente\_ID è semplice, perché in quella colonna della tabella CLIENTE il tipo di dati è standard.

E se si desidera selezionare i nomi delle persone dalla tabella CLIENTE? Non sarebbe possibile impostare una query come quella riportata di seguito:

```
select Nome /* non funziona */
  from CLIENTE;
```

perché Nome non è una colonna della tabella CLIENTE. Viene visualizzato il seguente messaggio di errore:

```
select Nome
*
ERROR at line 1:
ORA-00904: invalid column name
```

Oracle visualizza questo errore perché Nome non è una colonna della tabella CLIENTE. Nome infatti è un attributo nell'ambito del tipo di dati astratto PERSONA\_TY. Per visualizzare

i valori di Nome, occorre impostare una query per l'attributo Nome all'interno della colonna Persona. Non è possibile interrogare direttamente PERSON\_TY, perché si tratta semplicemente di un tipo di dati, non di una tabella. La struttura della query corretta è quella che compare nell'esempio seguente:

```
select Cliente_ID, C.Persona.Nome
  from CLIENTE C;
```

Si osservi la sintassi per la colonna Nome:

C.Persona.Nome

Innanzitutto occorre osservare che l'accesso all'attributo del tipo di dati richiede il ricorso a un *alias di tabella*; quest'ultimo, noto anche come *variabile di correlazione*, consente a Oracle di risolvere qualunque problema di ambiguità che riguardi il nome dell'oggetto selezionato.

Come nome di colonna, Persona.Nome indica l'attributo Nome nell'ambito del tipo di dati PERSONA\_TY. Il formato per il nome di colonna è:

Correlazione.Colonna.Attributo

Ciò può creare un po' di confusione. Nel comando insert viene utilizzato il nome del *tipo di dati* (in realtà si usa il nome del metodo costruttore, che equivale al nome del tipo di dati). Nel comando select viene invece utilizzato il nome della *colonna*. Ciò avviene perché potrebbero esserci più colonne che utilizzano il medesimo tipo di dati nella tabella.

Come si procede se si desidera utilizzare il comando select per selezionare i valori di Via dalla tabella CLIENTE? La colonna Via fa parte del tipo di dati INDIRIZZO\_TY, che a sua volta è parte del tipo di dati PERSONA\_TY. Per selezionare queste informazioni, occorre ampliare il formato Colonna.Attributo in modo da includere il tipo di dati annidato. Il formato corretto è il seguente:

Correlazione.Colonna.Colonna.Attributo

Quindi, per selezionare l'attributo dell'attributo INDIRIZZO all'interno della colonna Persona occorre impostare la query:

```
select C.Persona.Indirizzo.Via
  from CLIENTE;
```

PERSONA.INDIRIZZO.VIA

---

57 MT PLEASANT ST  
1 STEPAHEAD RD

La sintassi:

```
select C.Persona.Indirizzo.Via
```

indica a Oracle con esattezza dove trovare l'attributo Via. La variabile di correlazione può essere un nome qualunque di propria scelta (seguendo gli standard di denominazione delle tabelle utilizzati da Oracle), a patto che non entri in conflitto con il nome di qualunque altra tabella coinvolta nella query.

Se si utilizzano tipi di dati astratti, non è possibile né inserire (con il comando insert), né selezionare (con il comando select) valori per gli attributi dei tipi di dati astratti senza conoscere

l'esatta struttura degli attributi stessi. Per esempio, non è possibile selezionare i valori per Citta della tabella CLIENTE senza sapere che Citta fa parte dell'attributo Indirizzo e che quest'ultimo fa parte della colonna Persona. Non è possibile inserire (comando `insert`) o aggiornare (comando `update`) i valori di una colonna se non si conosce il tipo di dati di cui fa parte e la struttura di annidamento dei tipi di dati per arrivare a quei valori.

Come si procede se è necessario fare riferimento alla colonna Citta in una clausola `where`? Come nell'esempio precedente, si può indicare Citta come parte dell'attributo Indirizzo, annidato all'interno della colonna Persona, come mostrato di seguito:

```
select C.Persona.Nome,
       C.Persona.Indirizzo.Citta
  from CLIENTE
 where C.Persona.Indirizzo.Citta like 'F%';
```

PERSONA.NOME	PERSONA.INDIRIZZO.CITTA
NEIL MULLANE	FINN

Quando si aggiornano i dati nell'ambito di tipi di dati astratti, occorre fare riferimento agli attributi utilizzando la sintassi Colonna.Attributo nel modo mostrato negli esempi precedenti. Per esempio, per modificare il valore di Citta per i clienti che vivono a Briant, nel New Hampshire, occorre eseguire la procedura seguente con il comando `update`:

```
update CLIENTE C
   set C.Persona.Indirizzo.Citta = 'HART'
  where C.Persona.Indirizzo.Citta = 'BRIANT';
```

La clausola `where` viene utilizzata da Oracle per trovare i record corretti da aggiornare, mentre `set` serve per impostare i nuovi valori per le colonne Citta.

Come mostrato negli esempi citati, l'utilizzo di tipi di dati astratti semplifica la rappresentazione dei dati, ma complica il modo in cui occorre impostare le query e operare con i dati. È opportuno valutare i vantaggi dei tipi di dati astratti (rappresentazione dei dati più intuitiva) alla luce del potenziale aumento di complessità nelle procedure di accesso ai dati.

Negli ultimi capitoli del libro vengono descritti utilizzo e implementazione di caratteristiche orientate agli oggetti aggiuntive, come tabelle annidate e array variabili. Con i tipi di dati astratti si dispone di un punto di partenza comune per implementare caratteristiche orientate agli oggetti all'interno di un database Oracle. Nella Parte quarta vengono presentati ulteriori impieghi di tipi di dati astratti, tabelle annidate e array variabili.

## 4.5 Analisi e progettazione orientata agli oggetti

Nel Capitolo 2 sono stati illustrati i principi di base della normalizzazione: lo sviluppo di un'applicazione di database relazionale. Quando si considera la prospettiva di aggiungere caratteristiche orientate agli oggetti come i tipi di dati astratti, occorre un approccio alla progettazione del database con un'ottica leggermente diversa. Per esempio, nella normalizzazione tradizionale si cerca di collegare ciascun attributo alla sua chiave primaria. Nella progettazione orientata agli oggetti, occorre andare oltre la normalizzazione e localizzare gruppi di colonne che definiscano una rappresentazione dell'oggetto comune.

Per esempio, nella progettazione relazionale una tabella CLIENTE può essere creata in questo modo:

---

```
create table CLIENTE
(CLiente_ID NUMBER primary key,
Nome      VARCHAR2(25),
Via       VARCHAR2(50),
Citta     VARCHAR2(25),
Prov      CHAR(2),
Cap       NUMBER);
```

Ragionando in termini di terza forma normale, si tratta di una rappresentazione corretta della tabella CLIENTE. Ciascuno degli attributi è correlato unicamente al Cliente\_ID (la chiave primaria della tabella). Tuttavia, osservando la tabella si può verificare che sono presenti delle colonne che, quando vengono raggruppate insieme, rappresentano un oggetto. Negli esempi precedenti le colonne Via, Citta, Prov e Cap venivano raggruppate in un unico tipo di dati chiamato INDIRIZZO\_TY, tramite il quale è possibile utilizzare lo stesso tipo di dati in più tabelle. Se il tipo di dati astratto viene utilizzato per diverse tabelle relazionali, è possibile sfruttare il vantaggio del riutilizzo degli oggetti e dell'aderenza alla definizione standard delle strutture di attributi.

Una volta definiti i tipi di dati astratti, occorre cercare le relazioni tra di essi per verificare se devono essere posti in una struttura annidata (come nel caso di PERSONA\_TY e INDIRIZZO\_TY). Quando si analizzano le possibilità di impostazione dei tipi di dati per il proprio database, occorre focalizzare l'attenzione su quei tipi di dati che verranno riutilizzati o che hanno sempre lo stesso tipo di modalità operative. Una volta definiti i tipi di dati, si possono applicare nella creazione di nuovi oggetti di database. Se esistono già delle tabelle, è possibile utilizzare le viste oggetti per sovrapporre i modelli orientati agli oggetti alle tabelle relazionali. Per ulteriori informazioni sulle viste oggetti, si consulti la Parte quarta.

Dopo aver definito i tipi di dati, si possono creare dei metodi per ciascun tipo. I metodi si utilizzano per definire le modalità operative di accesso ai dati che coinvolgono i tipi di dati impostati. Solitamente i metodi vengono creati sulla base delle modalità di utilizzo dei dati, per ciascuna delle quali dovrebbero essere impostati dei metodi. Nella Parte quarta vengono illustrati alcuni esempi di metodi definiti dall'utente.

## 4.6 I prossimi capitoli

Con Oracle è possibile utilizzare un database come rigidamente relazionale, come relazionale orientato agli oggetti, o come orientato agli oggetti. Fondamentalmente, Oracle è un database relazionale; le caratteristiche orientate agli oggetti sono implementate come estensioni del motore relazionale. Per questo motivo occorre acquisire familiarità con le caratteristiche relazionali di Oracle, prima di iniziare a utilizzare caratteristiche orientate agli oggetti, e in questa parte del libro viene illustrato questo tipo di approccio. Nella parte seconda vengono descritte in maniera dettagliata le implementazioni in Oracle del linguaggio SQL. Questa parte del libro è imperniata sull'implementazione di SQL per tabelle relazionali; con certi limiti, le stesse funzioni sono adeguate per tabelle con caratteristiche orientate agli oggetti come i tipi di dati astratti. Dopo i capitoli sull'utilizzo di SQL, si trovano sezioni dedicate a PL/SQL, (l'estensione procedurale di Oracle a SQL) e a Java. Una volta acquisita familiarità con PL/SQL, l'utente sarà in grado di creare autonomamente metodi per i tipi di dati, come illustrato nei capitoli relativi alle modalità orientate agli oggetti. Quando si considera l'idea di utilizzare le caratteristiche avanzate, occorre ricordare che si tratta di estensioni di SQL; per il loro utilizzo efficace è quindi essenziale formarsi una solida base in questo campo.



## Capitolo 5

# Introduzione ai database con capacità Web

- 5.1 **Dove si inserisce SQL?**
- 5.2 **Dove si inserisce Java?**
- 5.3 **Dove si inserisce Oracle Portal?**

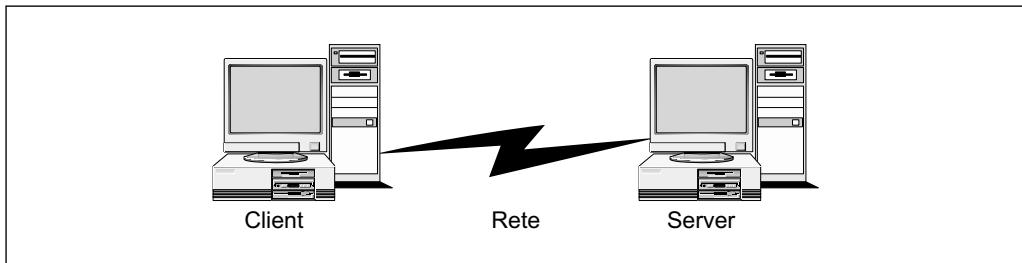
Molti esempi di questo libro sono impernati sull'impiego di SQL nel database. Gli esempi presuppongono la possibilità di connettersi direttamente a un database Oracle, oppure l'impiego di uno strumento client che permetta di inserire dei comandi in un database. Con l'introduzione di tecnologie Web, la possibilità che l'utente ha di interagire con i database si è arricchita di nuove opzioni. La conoscenza di SQL è sempre necessaria, tuttavia per sfruttare queste nuove opzioni potrebbe essere richiesta anche la conoscenza dei Web listener, di nuovi insiemi di strumenti, di linguaggi come Java e di interfacce per la programmazione di applicazioni (API) come JDBC e SQLJ.

Inizialmente, la complessità dell'ambiente potrebbe risultare scoraggiante. Si consideri un'architettura semplice, mostrata nella Figura 5.1, in cui due computer, un client e un server, sono coinvolti nell'applicazione. Il database si trova sul server e l'utente interagisce con quest'ultimo attraverso il client. Per le loro comunicazioni, il client e il server fanno affidamento sulla rete di base, e ciascuno esegue una versione di Oracle Net (noto nelle versioni precedenti come Net8). Il server ascolta le richieste Oracle Net inviate dal client.

Ora si consideri un'architettura a tre livelli, come quella della Figura 5.2. Un'architettura di questo tipo presenta tre componenti separati: un client, un application server e un database server. Implementando un'architettura a tre livelli, le opzioni disponibili sono molte di più di quelle offerte da un'architettura client-server tradizionale. Il protocollo impiegato nelle comunicazioni tra il client e l'application server può differire da quello utilizzato nelle comunicazioni tra l'application server e il database server. La distribuzione del carico di lavoro tra i tre componenti può variare moltissimo da un'applicazione all'altra. Le prestazioni e l'affidabilità dei componenti nella modalità standalone e in quella di rete possono influire sull'esito positivo o negativo dell'applicazione.

**NOTA** *In alcune implementazioni a tre livelli, l'application server e il database server si trovano sullo stesso server fisico.*

L'architettura a tre livelli aumenta significativamente le possibilità di distribuire il carico di lavoro di un'applicazione tra più server. Essa tuttavia aumenta anche il numero di possibili errori nell'architettura, e complica la configurazione di un ambiente di prova da utilizzare prima del passaggio al livello di produzione. Oracle offre molti modi diversi di implementare un'architettura a tre livelli, quindi si dovrà selezionare l'architettura che più di ogni altra soddisfa le esigenze dell'applicazione. Per ogni applicazione nuova si dovrebbero valutare nuovamente le esigenze ambientali, invece di partire dal presupposto che una singola architettura supporterà senza problemi qualsiasi risultato desiderato.



**Figura 5.1** Architettura client-server.

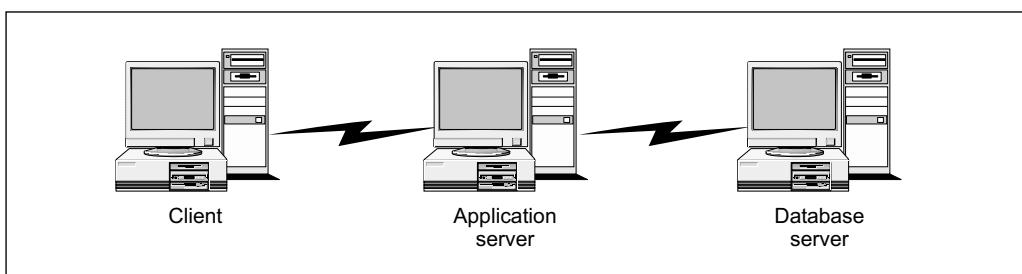
La maggior parte dei database con capacità Web si basa su un modello a tre livelli. In genere, un database server già esistente è abilitato all'accesso tramite Web. Per abilitare il database, il server dev'essere accessibile tramite una rete esterna. Per fornire questo accesso di rete, solitamente si ricorre a un secondo server utilizzato come firewall, il quale limita i tipi di comandi che possono essere passati al database server. L'application server può fungere da firewall.

Si consideri l'architettura illustrata nella Figura 5.3. Basandosi sulla Figura 5.2, questa immagine mostra una possibile configurazione di un database con capacità Web. Nella Figura 5.3, il client è un computer con accesso a Internet che esegue un browser. Questo client comunica con l'application server attraverso il protocollo HTTP (Hypertext Transfer Protocol). A sua volta, l'application server esegue dei comandi sul database, formatta i risultati in Hypertext Markup Language (HTML) e li restituisce al client.

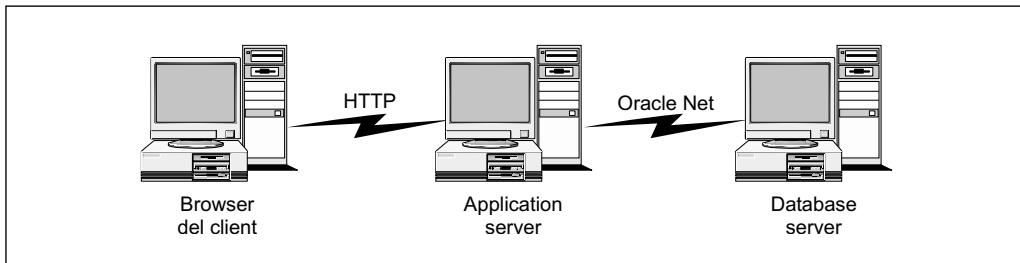
In questa configurazione, l'application server fornisce servizi di autenticazione (per assicurare che il client abbia il permesso di dare inizio alla richiesta), servizi di connessione al database e servizi di elaborazione delle applicazioni. Il ruolo del client consiste nell'inizializzare la richiesta e visualizzare i risultati restituiti, mentre il database funge da repository per i dati. Ovviamente, si dovrebbero limitare i privilegi di cui gode l'application server per garantire che solo i comandi autorizzati possano essere eseguiti per conto dei client.

## 5.1 Dove si inserisce SQL?

SQL è il linguaggio standard impiegato per accedere ai dati di Oracle, quindi dove lo si può inserire in un'architettura basata sul Web? SQL e il linguaggio procedurale di Oracle, PL/SQL, sono utilizzati comunemente in diverse situazioni.



**Figura 5.2** Architettura a tre livelli.



**Figura 5.3** Una comune implementazione Web.

- All'interno del database server, sotto forma di stored procedure e package (fare riferimento alla Parte terza).
- Nei comandi eseguiti per creare e mantenere le strutture di database.
- Nelle query dirette del database.
- Nei processi batch e negli script eseguiti sul database server.
- Nei comandi eseguiti dall'application server.

La propria architettura potrebbe prevedere altri casi in cui viene impiegato SQL, tuttavia quelli appena elencati sono gli esempi più comuni. Dato che il client e l'application server potrebbero comunicare senza ricorrere a SQL, è necessario che venga eseguito un Web listener sull'application server per ricevere le richieste HTTP provenienti dai client e iniziare a elaborarle. Se l'application server comunica con il database server attraverso SQL, richiederà anche la presenza di Oracle Net.

Invece di utilizzare il linguaggio SQL standard per comunicare con il database server, l'application server potrebbe impiegare Java Database Connectivity (JDBC) e SQLJ (un precompilatore che genera codice JDBC). Per ulteriori dettagli sulla creazione e l'utilizzo di programmi JDBC e SQLJ si rimanda alla Parte quinta. Se si utilizzano JDBC o SQLJ, sarà comunque necessario mettere a disposizione Oracle Net sull'application server per supportare la connettività al database. Sarà inoltre indispensabile una buona conoscenza di SQL e dei vari modi in cui JDBC e SQLJ offrono supporto per differenti insiemi di risultati.

## 5.2 Dove si inserisce Java?

Considerata l'onnipresenza di Java sul Web, si potrebbe cedere alla tentazione di creare un'applicazione per database basata esclusivamente su Java. In un'architettura di questo genere, il client comunica con l'application server tramite HTTP oppure tramite Internet Inter-ORB Protocol (IIOP). Nell'ambito del database è possibile creare classi Java, la cui interazione con l'application server avverrebbe tramite IIOP. Pertanto è possibile generare un database con capacità Web che non si basa su Oracle Net per la connettività.

La scrittura delle stored procedure all'interno del database può avvenire con PL/SQL (fare riferimento alla Parte terza), oppure con Java (consultare la Parte quinta). In linea di massima, questi linguaggi dovrebbero essere utilizzati per le funzioni che svolgono al meglio, ossia PL/SQL per l'interazione con il database e Java per le funzioni che non sono direttamente correlate al database. Se si usa esclusivamente Java in tutta l'applicazione, si dovrà mettere in conto l'eventualità di scontrarsi con limitazioni nelle prestazioni e nella funzionalità durante il suo utilizzo a livello di produzione.

Java è adatto per essere impiegato sul client e sull'application server. Durante la fase di sviluppo dell'applicazione, è consigliabile verificare le implicazioni per l'accesso al database derivanti dall'uso di Java al posto di PL/SQL o SQL. PL/SQL fa parte del kernel Oracle da più di dieci anni ed è particolarmente adatto per il recupero e la manipolazione rapida dei dati. Anche se dal punto di vista tecnico Java è applicabile a tutti i server visibili nella Figura 5.3, è comunque necessario valutare attentamente in che modo Java risponde alle proprie esigenze.

### 5.3 Dove si inserisce Oracle Portal?

Oracle Portal è un prodotto Oracle impiegato nella creazione di siti e applicazioni Web (noti nelle release precedenti come WebDB). I dati fondamentali per le applicazioni e i siti sono memorizzati in tabelle Oracle. Quando un client accede al sito, la sua richiesta viene gestita da un Web listener, come Oracle 9i Application Server (9iAS). Il Web listener esegue le funzioni PL/SQL memorizzate all'interno del database, le quali restituiscono i dati richiesti incorporati in tag HTML. Successivamente il listener restituisce questi dati al client e il sito viene visualizzato nel browser del client.

Il concetto di tabella, introdotto nel Capitolo 1, potrebbe essere utile per illustrare questo processo. Per visualizzare la tabella CLIMA nel browser di un client, si potrebbe creare un'applicazione Oracle Portal. La tabella CLIMA si troverà nel database sul database server. Per visualizzare i dati sotto forma di tabella, la query per l'application server incorporerà nell'output le tag per la formattazione delle tabelle HTML. Il risultato è un file HTML che contiene le tag incorporate per la formattazione delle tabelle HTML insieme ai dati recuperati dalle righe della tabella CLIMA.

Oracle Portal può essere utilizzato per diversi scopi, tuttavia prima di implementarlo sarebbe consigliabile considerare i risultati che si intendono raggiungere. Dato che Oracle Portal richiede un'architettura a tre livelli, è necessario verificare che questa architettura sia adatta al proprio ambiente e alle proprie capacità di elaborazione.

Un database con capacità Web offre spazio a tutte le tecnologie: PL/SQL nei package, SQL negli script, Java nelle classi, Oracle Portal come interfaccia utente e altri prodotti Oracle come tecnologie integranti. Vista la disponibilità di nuovi strumenti di sviluppo e componenti di integrazione, si dovrebbe considerare l'eventualità di integrarli nella propria architettura. L'integrazione dei metodi di accesso ai dati e delle tecnologie di presentazione dei medesimi dovrebbe semplificare il mantenimento e la progettazione delle applicazioni, offrendo nel contempo una piattaforma in grado di estendersi per soddisfare i requisiti futuri dell'applicazione.

Parte seconda

## **SQL E SQL\*PLUS**



## Capitolo 6

# Report e comandi fondamentali di SQL\*PLUS

- 6.1    **Creazione di un report semplice**
- 6.2    **Altre caratteristiche**
- 6.3    **Controllo dell'ambiente SQLPLUS**
- 6.4    **I fondamenti**

**S**QLPLUS viene generalmente considerato come una sorta di compilatore di report interattivo. Questo strumento utilizza SQL per ottenere informazioni dal database Oracle e offre la possibilità di creare report in quanto permette di gestire facilmente titoli, intestazioni di colonna, totali parziali e totali, riformattazione di numeri e testo e molto altro ancora. Inoltre, può essere utilizzato per modificare il database tramite i comandi *insert*, *update* e *delete* in SQL. SQLPLUS può anche essere impiegato come generatore di codice: con una serie di comandi si può creare dinamicamente un programma e quindi eseguirlo.

La maggior parte delle applicazioni di produzione adotta compilatori di report più avanzati, come report costruiti su parametri legati al Web. SQLPLUS viene utilizzato molto più comunemente per semplici query e report stampati. Per fare in modo che SQLPLUS formatti le informazioni nei report secondo il proprio gusto e le proprie esigenze è sufficiente qualche *comando* o parola chiave che dia istruzioni a SQLPLUS su come operare. Questi comandi sono elencati nella Tabella 6.1, mentre nel Capitolo 42 è possibile trovare esempi, spiegazioni dettagliate e caratteristiche aggiuntive per ciascuno di questi comandi.

In questo capitolo verrà presentato un semplice report compilato con SQLPLUS e verranno illustrate le caratteristiche utilizzate per crearlo. Se la creazione di un report risulta all'inizio un po' faticosa, non è il caso di preoccuparsi. Una volta provati i diversi passaggi della procedura, questi risulteranno semplici da comprendere e diventeranno presto familiari.

È possibile compilare report SQLPLUS mentre si lavora in maniera interattiva con SQLPLUS; ossia si possono digitare comandi riferiti a intestazioni di pagina, titoli di colonna, formattazione, interruzioni, totali e così via, per poi eseguire una query SQL, e SQLPLUS produrrà immediatamente il report formattato secondo le indicazioni specificate. Si tratta di un approccio interessante nel caso di risposte rapide a domande semplici che non hanno probabilità di essere riproposte. Accade più spesso tuttavia di dover generare report complessi che devono essere prodotti periodicamente, e soprattutto che sia necessario stamparli invece di visualizzarli semplicemente sullo schermo. Purtroppo, quando si esce da SQLPLUS, vengono immediatamente dimenticate tutte le istruzioni impostate. Se si utilizzasse SQLPLUS soltanto in questa modalità interattiva, l'esecuzione successiva dello stesso report significherebbe inserire nuovamente tutti i dati.

L'alternativa è molto semplice: digitare i comandi in un file linea per linea. SQLPLUS potrà poi leggere questo file come se si trattasse di uno script, ed eseguire i comandi come se venissero digitati. In effetti ciò equivale alla creazione di un programma di report senza ricorrere a un programmatore o ad un compilatore. Questo file viene creato con uno qualsiasi dei comuni editor disponibili, o anche (con alcune limitazioni) con un elaboratore di testi.

**Tabella 6.1** Comandi di base di SQLPLUS.

COMANDO	DEFINIZIONE
remark	Indica a SQLPLUS che le parole a seguire vanno trattate come commenti e non come istruzioni.
set headsep	Il separatore di titolo identifica il singolo carattere che indica a SQLPLUS di suddividere un titolo in due o più righe.
ttitle	Imposta il titolo superiore per ogni pagina di un report.
btitle	Imposta il titolo inferiore per ogni pagina di un report.
column	Fornisce a SQLPLUS una varietà di istruzioni su titolo, formato e trattamento di una colonna.
break on	Indica a SQLPLUS dove inserire degli spazi tra le sezioni di un report o dove interrompere per subtotali e totali.
compute sum	Indica a SQLPLUS di calcolare subtotali.
set linesize	Imposta il massimo numero di caratteri consentiti su qualsiasi riga del report.
set pagesize	Imposta il massimo numero di righe per pagina.
set newpage	Imposta il numero di righe vuote tra le pagine.
spool	Reindirizza in un file un report che normalmente verrebbe visualizzato sullo schermo, in modo che sia possibile stamparlo.
/**/	Contrassegna l'inizio e la fine di un commento in una query SQL. È simile a remark.
--	Contrassegna l'inizio di un commento in linea in una query SQL. Tutto quanto si trova dal contrassegno alla fine della riga è considerato un commento. È simile a remark.
set pause	Interrompe la visualizzazione a ogni schermata.
save	Salva la query SQL che si sta creando nel file indicato.
host	Invia i comandi al sistema operativo host.
start o @	Indica a SQLPLUS di eseguire le istruzioni salvate in un file.
edit	Esce da SQLPLUS ed entra in un editor specificato dall'utente.
define _editor	Indica a SQLPLUS il nome dell'editor specificato dall'utente.
exit o quit	Termina SQLPLUS.

L'editor non fa parte di Oracle. Questi editor sono disponibili in centinaia di varietà e ognuno ha il suo preferito. Oracle si è resa conto di questa situazione e ha deciso di permettere all'utente di scegliere l'editor da utilizzare invece di fornire un editor integrato e obbligare gli utenti a usarlo. Quando si è pronti per utilizzare il proprio editor, è necessario sospendere SQLPLUS, passare all'editor, creare o modificare il programma di report di SQLPLUS (detto anche *file di avvio*), quindi ritornare a SQLPLUS esattamente nel punto in cui lo si era lasciato ed eseguire quel report (Figura 6.1).

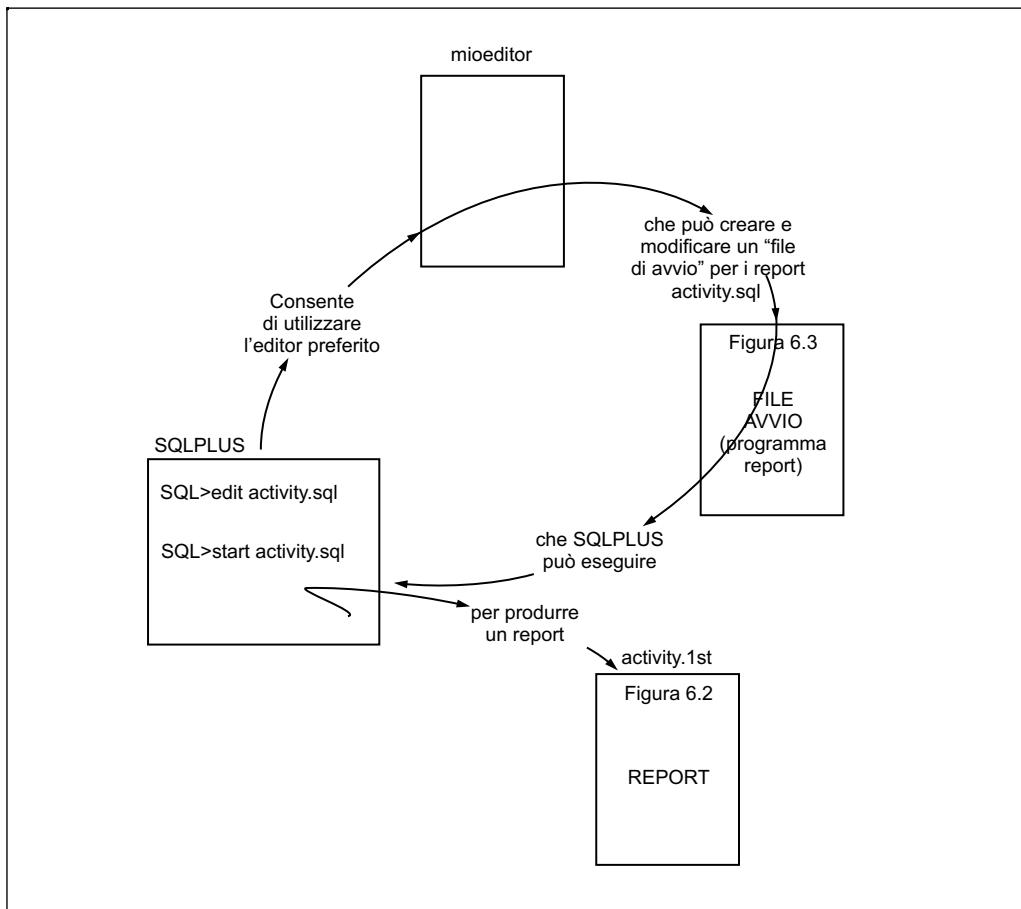


Figura 6.1 Processo di creazione dei report.

In SQLPLUS è disponibile anche un editor incorporato, talvolta definito *editor a linea di comando*, che consente di modificare rapidamente una query SQL senza uscire da SQLPLUS. L'uso di questa caratteristica verrà illustrato più avanti in questo capitolo.

## 6.1 Creazione di un report semplice

La Figura 6.2 illustra un report rapido e semplice che riporta le date in cui i libri sono stati presi in prestito e restituiti.

La Figura 6.3 illustra invece il file di avvio di SQLPLUS con il quale è stato prodotto questo report; in questo caso si tratta di activity.sql. Per eseguire questo programma di report in SQLPLUS, occorre digitare:

```
start activity.sql
```

Thu Apr 04				page 1
Registro prestiti dal 1/1/02 al 31/3/02				
NOME	TITOLO	DPRESTITO	DRESTITUZ	Giorni Fuori
DORAH TALBOT	EITHER/OR	02-JAN-02	10-JAN-02	8.00
	POLAR EXPRESS	01-FEB-02	15-FEB-02	14.00
	GOOD DOG, CARL	01-FEB-02	15-FEB-02	14.00
	MY LEDGER	15-FEB-02	03-MAR-02	16.00
*****				-----
avg				13.00
EMILY TALBOT	ANNE OF GREEN GABLES	02-JAN-02	20-JAN-02	18.00
	MIDNIGHT MAGIC	20-JAN-02	03-FEB-02	14.00
	HARRY POTTER AND THE	03-FEB-02	14-FEB-02	11.00
	GOBLET OF FIRE			
*****				-----
avg				14.33
FRED FULLER	JOHN ADAMS	01-FEB-02	01-MAR-02	28.00
	TRUMAN	01-MAR-02	20-MAR-02	19.00
*****				-----
avg				23.50
GERHARDT KENTGEN	WONDERFUL LIFE	02-JAN-02	02-FEB-02	31.00
	MIDNIGHT MAGIC	05-FEB-02	10-FEB-02	5.00
	THE MISMEASURE OF	13-FEB-02	05-MAR-02	20.00
	MAN			
*****				-----
avg				18.67
JED HOPKINS	INNUMERACY	01-JAN-02	22-JAN-02	21.00
	TO KILL A	15-FEB-02	01-MAR-02	14.00
	MOCKINGBIRD			
*****				-----
avg				17.50
PAT LAVAY	THE SHIPPING NEWS	02-JAN-02	12-JAN-02	10.00
	THE MISMEASURE OF	12-JAN-02	12-FEB-02	31.00
	MAN			
*****				-----
avg				20.50
ROLAND BRANDT	THE SHIPPING NEWS	12-JAN-02	12-MAR-02	59.00
	THE DISCOVERERS	12-JAN-02	01-MAR-02	48.00
	WEST WITH THE NIGHT	12-JAN-02	01-MAR-02	48.00
*****				-----
avg				51.67
				-----
avg				22.58
dalla Biblioteca				

Figura 6.2 Report dei prestiti dalla biblioteca.

## remark

La prima linea della Figura 6.3, contrassegnata dal numero 1, è la documentazione relativa al file di avvio. Le linee di documentazione iniziano con

```
rem
```

(abbreviazione di remark). SQLPLUS ignora qualsiasi cosa si trovi in una linea che inizia con queste lettere, il che consente di aggiungere commenti, annotazioni e spiegazioni a qualunque file di avvio creato. È sempre opportuno introdurre questi commenti all'inizio del file di avvio, indicando il nome del file, l'autore e la data di creazione, il nome di chi ha apportato modifiche, la data delle modifiche, gli elementi modificati e una breve spiegazione dello scopo del file. Si tratta di un accorgimento che si rivela indispensabile quando iniziano ad accumularsi decine di report.

```
rem Bookshelf activity report ———①
set headsep !———②
ttitle 'Registro prestiti dal 1/1/02 al 31/3/02'———③
btitle 'dalla Biblioteca'

column Nome format a20———④
column Titolo format a20 word_wrapped———⑤
column GiorniFuori format 999.99———⑥
column GiorniFuori heading 'Giorni!Fuori'———⑦

break on Nome skip 1 on report———⑧
compute avg of GiorniFuori on Nome———⑨
compute avg of GiorniFuori on report

set linesize 80———⑩
set pagesize 60
set newpage 0
set feedback off

spool activity.lst———⑪
select Nome, Titolo, DataPrestito, DataRestituzione,
       DataRestituzione-DataPrestito as GiorniFuori /*Count Days*/———⑫
  from BIBLIOTECA_PRESTITI
 order by Nome, DataPrestito;

spool off
```

**Figura 6.3** Il file activity.sql.

## Come distinguere SQLPLUS e SQL

**L**'istruzione di selezione visibile verso la fine della Figura 6.3, che inizia con il termine "select" e termina con un punto e virgola (;), è espressa in SQL, il linguaggio utilizzato per dialogare con il database Oracle. Ogni altro comando visibile nella pagina è un comando SQLPLUS, utilizzato per formattare i risultati di una query SQL in un report.

Il comando start fa in modo che SQLPLUS legga il file activity.sql ed esegua le istruzioni presenti in esso. Osservando con attenzione, ci si accorge che questo file di avvio contiene le istruzioni SQLPLUS di base da utilizzare per produrre report o modificare le modalità con cui si interagisce con SQLPLUS. In relazione all'esperienza di ciascuno, questo file può apparire avanzato o di livello elementare; è costituito da una serie di istruzioni semplici per SQLPLUS.

### set headsep

La punteggiatura al termine della linea set headsep ("heading separator", separatore di intestazione), contrassegnata dal numero 2 nella Figura 6.3, indica a SQLPLUS come verrà segnalato il punto in cui si intende interrompere un titolo di pagina o un'intestazione di colonna che siano più lunghi di una riga. La prima volta che si attiva SQLPLUS, il carattere headsep predefinito è una barra verticale ( | ), tuttavia se si preferisce usare le barre verticali nei titoli, nulla vieta di utilizzare un carattere headsep differente.

```
set headsep !
```

**ATTENZIONE** *Se si seleziona un carattere che può apparire all'interno di un titolo o di un'intestazione di colonna, può accadere che l'intestazione venga suddivisa in modo imprevisto.*

### ttitle e btitle

Nella linea contrassegnata dal numero 3 nella Figura 6.3:

```
ttitle 'Registro prestiti dal 1/1/02 al 31/3/02'
```

vengono date istruzioni a SQLPLUS per impostare questo titolo di testa (top title) all'inizio di ogni pagina del report. Il titolo scelto dovrà essere racchiuso tra apici. Questa linea:

```
btitle 'dalla Biblioteca'
```

funziona in maniera analoga alla precedente con ttitle, tranne per il fatto che il titolo deve essere posto in fondo alla pagina ("b" sta per "bottom", in fondo); anche questa notazione dev'essere racchiusa tra apici. Dato che gli apici servono per racchiudere il titolo intero, un apostrofo (che sulla tastiera corrisponde allo stesso carattere) ingannerebbe SQLPLUS facendogli credere erroneamente che il titolo è finito.

**NOTA** *Per usare gli apostrofi nei titoli, è possibile digitare due apici successivi quando si desidera inserire un apice. Dato che sia per SQL sia per SQLPLUS gli apici racchiudono stringhe di caratteri, questa tecnica viene utilizzata per entrambi ogni volta che si deve stampare o visualizzare un apostrofo.*

Quando si utilizza ttitle nel modo illustrato, SQLPLUS centrerà sempre il titolo scelto in base alle dimensioni di linea impostate (linesize verrà trattato più avanti in questo capitolo) e

inserirà sempre nell'angolo superiore sinistro il giorno della settimana, il mese e il giorno del mese in cui il report è stato eseguito, e nell'angolo superiore destro il numero di pagina.

I comandi `repheader` e `repfooter` possono essere utilizzati per creare le intestazioni e i piedi di pagina dei report. Per le descrizioni di `repheader` e `repfooter`, si rimanda al Capitolo 42.

## column

Con `column` è possibile modificare l'intestazione e il formato di qualsiasi colonna contenuta in un'istruzione `select`. Si osservi il report rappresentato nella Figura 6.2. La quinta colonna, "Giorni fuori", non è una colonna del database e prende il nome di `GiorniFuori` nella query illustrata nella Figura 6.3. La linea:

```
column GiorniFuori heading 'Giorni!Fuori'
```

assegna alla colonna una nuova etichetta e una nuova intestazione. Questa intestazione si divide in due linee perché in essa è incorporato il carattere `headsep` (!) La linea contrassegnata dal numero 4:

```
column Nome format a20
```

imposta la larghezza di visualizzazione della colonna `Nome` a 20. La "a" in "a20" indica a SQLPLUS che si tratta di una colonna di caratteri alfabetici e non numerici. La larghezza può essere impostata praticamente a qualsiasi valore, senza considerare come è stata definita la colonna nel database.

La colonna `Nome` è definita con ampiezza pari a 25 caratteri, quindi è possibile che qualche nome abbia più di 20 caratteri. Se nella definizione di questa colonna nel report non è stato specificato nient'altro, qualsiasi nome con più di 20 caratteri andrà a capo alla riga successiva. La colonna `Titolo` è stata definita come `VARCHAR2(100)`, tuttavia è formattata come a20 (si veda il numero 5).

Invece di utilizzare il formato `word_wrapped`, si potrebbe scegliere `truncated`, che non visualizza i caratteri che superano la lunghezza di formato specificata per la colonna.

Al punto 6 della Figura 6.3 viene mostrato un esempio di formattazione di un numero:

```
column GiorniFuori format 999.99
```

In questo modo, la colonna è stata definita per contenere cinque cifre e un punto decimale. Se però si contano gli spazi nel report per la colonna `GiorniFuori`, si osserverà che sono sette. Se invece si considera soltanto il comando `column`, si è portati a credere che la colonna debba avere una larghezza pari a sei spazi. In questo modo però non vi sarebbe spazio per un eventuale segno meno, nel caso in cui il numero sia negativo; per questo motivo con i numeri viene sempre previsto uno spazio in più a sinistra.

Nel punto 7 della Figura 6.3 si fa riferimento a una colonna che non appariva nella tabella quando è stato chiesto a SQLPLUS di descriverla:

```
column GiorniFuori heading 'Giorni!Fuori'
```

Che cosa significa `GiorniFuori`? Si osservi l'istruzione `select` nella parte inferiore della Figura 6.3. `GiorniFuori` appare nella linea:

```
DataRestituzione-DataPrestito as GiorniFuori /*Conta i giorni*/
```

che indica a SQL di eseguire calcoli aritmetici sulle date, ossia di contare il numero di giorni trascorsi tra due date e di assegnare a questo calcolo un nome di colonna più semplice. Di conseguenza, SQLPLUS visualizza una colonna di nome GiorniFuori e tutte le opzioni di formattazione e gli altri comandi sono attivi come se si trattasse di una vera colonna della tabella. Il comando column per GiorniFuori ne è un esempio. “GiorniFuori” viene indicato come un *alias di colonna*, ossia un altro nome da utilizzare quando si fa riferimento alla colonna.

## **break on**

Si osservi il punto 8 nella Figura 6.3. Nel report della Figura 6.2, si noti che i record estratti per ogni Nome sono stati raggruppati insieme. Questo effetto è stato ottenuto con la linea:

```
break on Nome skip 1 on report
```

e con la linea:

```
order by Nome, DataPrestito;
```

dell’istruzione select verso la fine del file di avvio.

SQLPLUS considera ciascuna linea così come viene riportata da Oracle e tiene traccia del valore che appare in Nome. Per le prime quattro linee, questo valore è “DORAH TALBOT”, per cui SQLPLUS visualizza le linee così come sono, mentre nella quinta linea, Nome cambia in EMILY TALBOT. SQLPLUS applica le istruzioni relative all’interruzione: nel caso in cui Nome cambi, la visualizzazione normale linea dopo linea dev’essere modificata saltando una linea. Tra le sezioni Nome contenute nel report comparirà una linea. A meno che i nomi non siano stati raggruppati per mezzo della clausola order by, non avrebbe senso saltare una linea ogni volta che Nome cambia impostando break on. Ecco il motivo per cui il comando break on e la clausola order by devono essere impiegati in maniera coordinata.

Inoltre, si può notare che DORAH TALBOT compare solamente nella prima riga della propria sezione, come tutti gli altri nomi. Questo serve per eliminare la ripetizione di ciascun nome in ogni riga di ogni sezione, che risulta visivamente poco piacevole. Volendo, è invece possibile visualizzare un nome in tutte le righe della propria sezione modificando il comando break on in questo modo:

```
break on Nome duplicate skip 1
```

L’output del report nella Figura 6.2 mostra una media di GiorniFuori per l’intero report. Per ottenere il totale complessivo di un report occorre aggiungere un’altra interruzione utilizzando il comando break on report. Si raccomanda di prestare attenzione quando si aggiungono delle interruzioni, poiché devono essere create tutte con un unico comando; se si inseriscono due comandi break on consecutivi, le istruzioni del primo comando vengono sostituite dal secondo. Si osservi il punto 8 per vedere il comando break on utilizzato nel report:

```
break on Nome skip 1 on report
```

## **compute avg**

Le medie calcolate per ciascuna sezione del report sono state prodotte con il comando compute avg evidenziato al punto 9. Questo comando funziona sempre congiuntamente al comando break on e i totali calcolati sono sempre riferiti alla sezione specificata da tale comando. Può quindi essere opportuno considerare questi due comandi correlati come una singola unità:

---

```
break on Nome skip 1 on report
compute avg of GiorniFuori on Nome
compute avg of GiorniFuori on report
```

In altre parole, si indica a SQLPLUS di calcolare la media di GiorniFuori per ciascun Nome. SQLPLUS calcolerà questo valore prima per DORAH TALBOT, poi per tutti i nomi successivi. Ogni volta che SQLPLUS rileva un nuovo Nome, calcola e visualizza una media per i valori di GiorniFuori precedenti. Inoltre, il comando `compute avg` inserisce una riga di asterischi sotto la colonna utilizzata in quel momento da `break on` e visualizza la parola “avg” al di sotto di essa. Per quei report in cui è necessario aggiungere parecchie colonne, per ogni calcolo si usa un’istruzione `compute avg` separata (oppure `compute sum`, se si calcolano i totali). In un report lungo (per Nome, Titolo, date, per esempio) è anche possibile avere molti tipi diversi di interruzioni, insieme a comandi `compute avg` correlati.

Si può utilizzare un comando `break on` anche senza `compute sum`, come nel caso in cui si debba organizzare il proprio report in sezioni in cui non è necessario impostare dei totali (per esempio, indirizzi con `break on` per Città), tuttavia non è vero il contrario.

**NOTA** *Ogni comando `compute avg` deve essere guidato da un comando `break on`, e la porzione `on` di entrambi i comandi deve corrispondere (come `on Nome` nell’esempio precedente, `break on Nome skip 1 on report` e `compute avg of GiorniFuori on Nome`).*

Ecco le regole fondamentali:

- Per ogni comando `break on` deve esistere un comando `order by` correlato.
- Per ogni comando `compute avg` deve esistere un comando `break on` correlato.

È facile comprendere, ma è altrettanto facile dimenticare qualche pezzo. Oltre a `compute avg`, si possono eseguire anche `compute sum`, `compute count`, `compute max`, oppure calcolare una qualsiasi delle altre funzioni di raggruppamento di Oracle sul set dei record.

## set linesize

I quattro comandi al punto 10 della Figura 6.3 controllano le dimensioni generali del report. Con `set linesize` si imposta il numero massimo di caratteri che possono comparire in una singola riga. Per documenti delle dimensioni di una lettera, questo numero è generalmente intorno a 70 o 80, a meno che la stampante non utilizzi un tipo di carattere molto compresso.

Se in una query SQL si inseriscono più colonne di informazioni di quante siano state stabilite con il parametro `linesize`, le colonne in sovrappiù vengono collocate sulla riga successiva e ammazzate una sotto l’altra. In realtà, si può utilizzare questa disposizione ottenendo un buon effetto quando è necessario presentare numerosi dati.

Il comando `linesize` serve anche per determinare il punto in cui dev’essere centrato `title` e quello in cui collocare la data e la numerazione di pagina. Questi due dati compaiono entrambi nella prima riga e la distanza tra la prima lettera della data e l’ultimo numero della numerazione di pagina deve sempre essere equivalente al valore di `linesize` impostato.

## set pagesize

Con il comando `set pagesize` si imposta il numero totale di righe che SQLPLUS inserirà in ogni pagina, compresi `title`, `btitle`, le intestazioni di colonna e qualsiasi eventuale riga vuota. Per il formato da lettera e quello generalmente utilizzato per i computer, questo numero solitamente è

66 (6 linee per pollice, moltiplicato per 11 pollici). Il comando `set pagesize` è correlato al comando `set newpage`.

### **set newpage**

Una definizione migliore per il comando `set newpage` (che in inglese significa “imposta nuova pagina”) avrebbe potuto essere “imposta righe vuote”, in quanto questo comando visualizza delle righe vuote prima della prima riga (quella in cui compaiono data e numerazione di pagina) di ogni pagina del report. Si tratta di una funzione utile sia per adattare la posizione dei report che vengono stampati su pagine singole con una stampante laser, sia per saltare le perforazioni tra le pagine tipiche della carta per moduli continui.

**NOTA** *Con questo comando non vengono impostate le dimensioni del corpo del report (il numero di linee visualizzate dalla data fino al btitle); viene impostata la lunghezza totale della pagina, misurata in righe.*

Pertanto, se si digita:

```
set pagesize 66  
set newpage 9
```

SQLPLUS produrrà un report che inizia con 9 righe vuote, seguite da 57 righe di informazioni (contando dalla data fino al btitle). Se si aumenta la dimensione di `newpage`, SQLPLUS visualizzerà meno linee di informazioni su ciascuna pagina, producendo però più pagine insieme.

Tutto ciò è facilmente comprensibile, ma che cosa è successo al punto 10 della Figura 6.3? In questo punto compare:

```
set pagesize 60  
set newpage 0
```

Si tratta di una dimensione insolita per la pagina di un report. L’istruzione di inserire zero righe vuote tra le pagine è stata impostata da SQLPLUS? No. Il numero zero dopo `newpage` attiva una proprietà speciale: il comando `set newpage 0` produce un *carattere di inizio pagina* (generalmente hex 13) proprio prima della data su ciascuna pagina. Per la maggior parte delle stampanti moderne ciò equivale a uno spostamento immediato all’inizio della pagina successiva, dove inizierà la stampa del report. La combinazione dei due comandi `set pagesize 60` e `set newpage 0` ha per effetto la produzione di un report il cui corpo di informazioni è lungo esattamente 60 righe, con un carattere di inizio pagina all’inizio di ogni pagina. Si tratta di un modo più semplice per controllare la stampa delle pagine rispetto al destreggiarsi tra righe vuote e numero di linee per pagina. È anche possibile utilizzare il comando `set newpage none`, con il risultato di non avere nessuna linea vuota e nessun carattere di inizio pagina tra le pagine del report.

### **spool**

Ai primordi dell’era informatica, la memorizzazione dei file avveniva nella maggior parte dei casi su bobine di filo magnetico o nastro. Scrivere informazioni su un file ed eseguire lo spooling di un file erano praticamente la stessa cosa. Il termine “*spooling*” è comunque sopravvissuto e

oggi si riferisce in genere a qualsiasi processo di spostamento di informazioni da un luogo a un altro. In SQLPLUS,

```
spool activity.lst
```

indica a SQL di prendere tutto l'output da SQLPLUS e trascriverlo nel file activity.list. Una volta inserito questo comando, l'operazione viene effettuata finché non si digita il comando opposto, ovvero:

```
spool off
```

Ciò significa, per esempio, che si potrebbe digitare:

```
spool work.fil
```

e quindi inserire una query SQL, come:

```
select Argomento, Sezione, Pagina from GIORNALE
  where Sezione = 'F';
```

ARGOMENTO	S	PAGINA
Nascite	F	7
Annunci	F	8
Necrologi	F	6
Salute	F	6

oppure una serie di comandi SQLPLUS come:

```
set pagesize 60
column Sezione heading 'I miei preferiti'
```

o qualsiasi altra cosa. Qualunque prompt prodotto da SQLPLUS, qualunque messaggio di errore, qualsiasi cosa compaia sullo schermo del computer durante l'operazione di spooling verrà trasferita nel file work.fil. L'operazione di spooling non fa distinzioni. Registra tutto quello che accade dal momento in cui si utilizza il comando spool al momento in cui si utilizza spool off, il quale ci riporta al report indicato al punto 11 della Figura 6.3:

```
spool activity.lst
```

Questa espressione viene collocata con attenzione come comando immediatamente prima dell'istruzione select, dopo la quale compare immediatamente il comando spool off. Se il file spool activity.lst fosse apparso un po' prima, i comandi SQLPLUS eseguiti sarebbero andati a finire nella prima pagina del file di report. Al contrario, questi vanno a finire nel file activity.lst, ossia ciò che compare nella Figura 6.2: i risultati della query SQL, formattati secondo le istruzioni date, e niente altro. Ora è possibile stampare tranquillamente il file, fiduciosi che dalla stampante uscirà un report formattato in modo chiaro.

Questo gruppo di comandi visualizzerà la query SQL sulla prima pagina dell'output, seguita dai dati che inizieranno nella seconda pagina. Per non visualizzare la query SQL insieme all'output, si può anche modificare l'ordine dei comandi: inserire la query SQL senza il punto e virgola finale. Premere due volte INVIO e il comando sarà ancora nel buffer di SQLPLUS, senza essere stato eseguito. Quindi, si potrà iniziare l'operazione di spooling e l'esecuzione del comando:

```
(SQL command typed here)
```

```
spool activity.lst
/
spool off
```

```
/* */
```

Il punto 12 della Figura 6.3 mostra come incorporare un commento in un'istruzione SQL. Si tratta di un metodo e di un tipo di utilizzo differente dall'istruzione remark discussa in precedenza. Il termine **remark** (o **rem**) deve apparire all'inizio di una linea e funziona soltanto per la linea in cui compare. Inoltre, in un'istruzione SQL a più linee non è consentito inserire **remark**. Quindi:

```
select Argomento, Sezione, Pagina
rem questo è solo un commento
  from GIORNALE
where Sezione = 'F';
```

non è corretto. Non funzionerà, e verrà visualizzato un messaggio di errore. Tuttavia, è possibile incorporare dei commenti in un comando SQL seguendo il metodo illustrato al punto 12, oppure come in questo esempio:

```
select Argomento, Sezione, Pagina
/* questo è solo un commento */
  from GIORNALE
where Sezione = 'F';
```

Il segreto sta nel sapere che il segno `/*` indica a SQLPLUS che è iniziato un commento. Qualsiasi cosa compaia da questo punto in poi, anche se prosegue con molte parole per diverse linee, viene considerata come un commento fino all'inserimento del segno `*/`, che indica la fine del commento in questione. Per iniziare un commento si possono anche utilizzare i caratteri `--` (due trattini). In questo caso, la fine della linea indica la fine del commento. Questo tipo di commento funziona come la versione a una linea di `/* */`, tranne per il fatto che viene utilizzato il segno `--` (due trattini).

## Qualche chiarimento sulle intestazioni di colonna

È possibile che la differenza tra la nuova denominazione che si nota in:

```
DataRestituzione-DataPrestito as GiorniFuori
```

e la nuova intestazione assegnata alla colonna Articolo in:

```
column GiorniFuori heading 'Giorni!Fouri'
```

non risulti affatto chiara, in particolare se si osserva il comando seguente:

```
compute avg of GiorniFuori on Nome
```

I comandi SQLPLUS conoscono solo le colonne che compaiono effettivamente nell'istruzione `select`. Ogni comando `column` è riferito a una colonna contenuta nell'istruzione `select`. Sia

break on, sia compute sono riferiti esclusivamente alle colonne presenti in questa istruzione. L'unica spiegazione per cui un comando column o compute sappia dell'esistenza della colonna GiorniFuori è che abbia ottenuto questa denominazione nell'istruzione select. La nuova denominazione di "DataRestituzione-DataPrestito" in "GiorniFuori" viene effettuata da SQL, *non* da SQLPLUS.

## 6.2 Altre caratteristiche

Non è così difficile osservare il contenuto di un file di avvio e il report generato per individuare quali modalità di formattazione e di calcolo sono state applicate. È possibile iniziare creando il file di avvio, inserendo in esso ciascuno dei comandi ritenuti necessari, eseguendolo infine in SQLPLUS per verificare che sia corretto. Tuttavia, la prima volta che si creano dei report, risulta spesso più semplice operare in modo interattivo con SQLPLUS, modificando i formati di colonna, le query SQL, i titoli e i totali finché ciò che si ha in mente inizia a prendere forma.

### Editor a linea di comando

Quando si digita un'istruzione SQL, SQLPLUS ricorda ogni linea così come è stata inserita, memorizzandola nel *buffer SQL* (un nome strano per indicare la memoria scratchpad di un computer in cui vengono memorizzate le istruzioni SQL). Si supponga di aver digitato questa query:

```
select Argometno, Sezione, Pagina
      from GIORNALE
     where Sezione = 'F';
```

SQLPLUS risponde nel seguente modo:

```
select Argometno, Sezione, Pagina
      *
ERROR at line 1: ORA-0704: invalid column name
```

Questo messaggio fa capire che la parola "Argomento" è stata digitata in modo errato. Nonostante ciò, non sarà necessario digitare nuovamente l'intera query. L'editor a linea di comando è già pronto in attesa di istruzioni. Innanzi tutto, si chieda di visualizzare la query:

```
list
```

Viene immediatamente visualizzata la risposta:

```
1  select Argometno, Sezione, Pagina
2  from GIORNALE
3* where Sezione = 'F'
```

Si osservi che SQLPLUS visualizza tutte e tre le linee numerandole. Inoltre, inserisce un asterisco accanto alla linea 3, per indicare la linea che si può modificare. Si desidera però modificare la linea 1, per cui sarà necessario digitare, e SQLPLS dovrà visualizzare, quanto segue:

```
list 1
```

```
1* select Argometno, Sezione, Pagina
```

Viene visualizzata la linea 1 come linea corrente. Ora, questa linea potrà essere modificata digitando:

```
change /Argomento/Argomento  
1* select Argomento, Sezione, Pagina
```

Si può verificare nuovamente l'intera query con:

```
list  
1 select Argomento, Sezione, Pagina  
2 from GIORNALE  
3* where Sezione = 'F'
```

Se il test è positivo, inserire una barra dopo il prompt. Questa barra non ha nulla a che fare con il comando change o con l'editor. Al contrario, comunica a SQLPLUS di eseguire il codice SQL memorizzato nel buffer.

```
/  
  
ARGOMENTO      S  PAGINA  
----- -  -----  
Nascite        F    7  
Annunci        F    8  
Necrologi      F    6  
Salute         F    6
```

Per eseguire il comando change è necessario segnalare l'inizio e la fine del testo che deve essere modificato con una barra (/) o con altri caratteri. La linea:

```
change $Argomento$Argomento
```

avrebbe ottenuto lo stesso risultato. SQLPLUS considera il primo carattere dopo la parola “change” come quello prescelto per segnalare l’inizio e la fine del testo da correggere (questi indicatori in genere vengono definiti *delimitatori*). È anche possibile cancellare la linea corrente, come mostrato di seguito:

```
list  
1 select Argomento, Sezione, Pagina  
2 from GIORNALE  
3* where Sezione = 'F'  
  
del  
  
list  
1 select Argomento, Sezione, Pagina  
2 from GIORNALE
```

del cancellerà solo ciò che si trova nella linea corrente. Per cancellare più linee contemporaneamente, è possibile impostare un intervallo di numeri di linea da cancellare con il comando del, specificando il primo e l’ultimo numero di linea per l’intervallo di linee da cancellare. Per cancellare le linee dalla 3 alla 7, occorre utilizzare del 3 7. Si osservi che in questo comando c’è uno spazio prima del numero della prima linea da cancellare (3) e un altro prima del numero

dell'ultima linea da cancellare (7). Se non si introducono spazi tra il “3” e il “7”, SQLPLUS cercherà di cancellare la linea 37. Per eliminare le linee dalla 2 fino alla fine del buffer, occorre usare del 2 LAST.

Scrivendo lettera per lettera la parola “delete” (cancellare) vengono eliminate tutte le linee e questa parola viene inserita alla linea 1. Ciò causerà solo dei problemi, quindi è consigliabile evitare di digitare l'intera parola “delete”. Se lo scopo è cancellare completamente l'istruzione select, è meglio digitare:

```
clear buffer
```

Se si desidera aggiungere qualcosa alla linea corrente, è possibile utilizzare il comando append:

```
list 1
```

```
1* select Argomento, Sezione, Pagina
```

```
append "Dovesta"
```

```
1* select Argomento, Sezione, Pagina "Dovesta"
```

con append il testo indicato viene collocato esattamente alla fine della linea corrente, senza lasciare nessuno spazio. Per inserire uno spazio, come è stato fatto in questo caso, occorre digitare *due* spazi tra la parola append e il testo.

È anche possibile inserire un'intera linea nuova dopo quella corrente utilizzando il comando input, come nell'esempio seguente:

```
list
```

```
1 select Argomento, Sezione, Pagina "Dovesta"
2*   from GIORNALE
```

```
input where Sezione = 'A'
```

```
list
```

```
1 select Argomento, Sezione, Pagina "Dovesta"
2   from GIORNALE
3* where Sezione = 'A'
```

quindi impostare l'intestazione di colonna per la colonna DoveSta:

```
column DoveSta heading "Dove sta"
```

e poi eseguire la query:

```
/
```

ARGOMENTO	S	Dove sta
Notizie	A	1
Editoriali	A	12

Riassumendo: l'editor a linea di comando può visualizzare l'istruzione SQL digitata con il comando list, modificare o cancellare la linea corrente (contrassegnata da un asterisco) con i

comandi change o delete, aggiungere qualcosa alla fine della linea corrente con append, oppure inserire un'intera linea dopo la linea corrente con input. Una volta apportate le correzioni necessarie, l'istruzione SQL verrà eseguita se si inserisce una barra al prompt SQL. Tutti questi comandi possono essere abbreviati digitando soltanto la prima lettera, tranne del, che dev'essere digitato esattamente con le tre lettere.

L'editor a linea di comando può modificare solamente l'istruzione SQL, non i comandi SQLPLUS. Se per esempio è stata digitata l'istruzione column Name format a18 e si desidera modificarla in column Name format a20, occorre digitare nuovamente l'intero comando (questo vale se si opera in modalità interattiva con SQLPLUS; se invece i comandi sono in un file, naturalmente li si potrà modificare con l'editor). Inoltre, si osservi che, in modalità interattiva, una volta iniziata la digitazione di un'istruzione SQL, è necessario completarla prima di poter inserire qualsiasi altro comando SQLPLUS, come i formati column o ttitle. Non appena SQLPLUS visualizza la parola select, considera tutto ciò che segue questa parola come parte dell'istruzione select finché non viene inserito un punto e virgola (;) alla fine dell'ultima linea dell'istruzione SQL, oppure una barra (/) all'inizio della linea successiva all'ultima dell'istruzione SQL.

Queste istruzioni sono entrambe corrette:

```
select * from REGISTRO;
```

```
select * from REGISTRO  
/
```

Questa invece non lo è:

```
select * from REGISTRO/
```

## **set pause**

Nello sviluppo di un nuovo report, oppure quando si utilizza SQLPLUS per query semplici del database, è generalmente utile impostare il parametro linesize a 79 o a 80, pagesize a 24 e newpage a 1. Occorre accompagnare queste impostazioni con due comandi correlati, come mostrato di seguito:

```
set pause 'Ancora...'  
set pause on
```

Con questa combinazione si ottiene una schermata piena di informazioni per ciascuna pagina del report prodotta, e la visualizzazione viene interrotta a ogni pagina per dare la possibilità di consultarla (la dicitura "Ancora..." apparirà nell'angolo in basso a sinistra) fino a quando non si preme INVIO. Dopo l'impostazione dei vari titoli e intestazioni di colonna, è possibile modificare nuovamente le dimensioni della pagina con pagesize in modo da adattarle a una pagina standard ed eliminare la pausa in questo modo:

```
set pause off
```

## **save**

Se le modifiche da apportare all'istruzione SQL sono ampie, o se semplicemente si desidera lavorare con il proprio editor, è necessario salvare le istruzioni SQL create fino a quel momento in modalità interattiva, trasferendole in un file, come questo:

---

```
save fred.sql
```

La risposta di SQLPLUS è la seguente:

```
Created file fred.sql
```

Le istruzioni SQL (ma non i comandi column, ttitle o altri comandi SQLPLUS) si trovano in un file di nome fred.sql (o un nome di propria scelta), che può essere modificato con il proprio editor.

Se il file esiste già, sarà necessario utilizzare l'opzione replace (abbreviata in rep) del comando save per salvare la nuova query in un file con quel nome. Per questo esempio, la sintassi sarebbe:

```
save fred.sql rep
```

## **store**

Il comando store può essere utilizzato per salvare in un file le impostazioni ambientali correnti di SQLPLUS. L'istruzione che segue creerà un file di nome my\_settings.sql e vi memorizzerà le impostazioni:

```
store set my_settings.sql create
```

Se il file my\_settings.sql dovesse già esistere, si potrebbe usare l'opzione replace al posto di create, e sostituire il vecchio file con le nuove impostazioni. In alternativa si potrebbe usare l'opzione append per aggiungere le impostazioni nuove a un file già esistente.

## **Editing**

Ognuno ha il proprio editor preferito. Con SQLPLUS è anche possibile utilizzare programmi di elaborazione di testi, ma soltanto se si salvano i file creati con essi in formato ASCII (si consulti il manuale del proprio elaboratore di testi per i dettagli relativi a questa operazione). Gli editor sono essi stessi dei programmi, che in genere vengono invocati digitando il loro nome al prompt del sistema operativo. Quello che segue è un comando tipico dell'ambiente UNIX:

```
> vi fred.sql
```

Nell'esempio, vi è il nome dell'editor e fred.sql rappresenta il file che si intende modificare (il file di avvio descritto in precedenza viene utilizzato in questo contesto soltanto come esempio; naturalmente occorre digitare il nome del file che si desidera effettivamente modificare). In altri tipi di computer non appare necessariamente il prompt, ma in ogni caso qualcosa di equivalente. Se è possibile richiamare un editor in questo modo sul proprio computer, quasi certamente lo si potrà fare dall'interno di SQLPLUS, *salvo che* non si digiti il nome dell'editor, ma il comando edit:

```
SQL> edit fred.sql
```

Per prima cosa, si dovrebbe indicare a SQLPLUS il nome dell'editor. Occorre pertanto definire l'editor in SQLPLUS, in questo modo:

```
define _editor = "vi"
```

## Uso di LOGIN.SQL per definire l'editor

**P**er definire il proprio editor con SQLPLUS occorre inserire il comando define\_editor in un file denominato: si tratta di un file speciale, che SQLPLUS ricerca a ogni avvio. Se lo trova, esegue tutti i comandi in esso contenuti come se fossero stati digitati manualmente; innanzi tutto cerca nella directory in cui l'utente si trova al momento di digitare SQLPLUS; se non lo trova, ricerca il file nella directory home di Oracle; se non lo trova nemmeno lì, interrompe la ricerca. Nel file login.sql può essere inserito qualsiasi comando utilizzabile in SQLPLUS, inclusi i comandi SQLPLUS e le istruzioni SQL; tutti vengono eseguiti prima di visualizzare il prompt SQL>. Questo può essere un modo pratico di impostare un proprio ambiente SQLPLUS, con tutte le opzioni preferite. Ecco l'esempio di un tipico file login.sql:

```
prompt Login.sql loaded.
set feedback off
set sqlprompt 'E ora, capo? '
set sqlnumber off
set numwidth 5
set pagesize 24
set linesize 79
define _editor="vi"
```

Un altro file denominato glogin.sql viene utilizzato per stabilire le impostazioni predefinite di SQLPLUS per tutti gli utenti di un database. Questo file, che in genere si trova nella directory amministrativa di SQLPLUS, è utile per forzare impostazioni di colonna e di ambiente per più utenti.

Il significato di tutti questi comandi è riportato nel Capitolo 42.

Si osservi che è stato inserito un trattino di sottolineatura prima della “e” di editor. SQLPLUS ricorderà quindi il nome dell’editor (finché non si esce da SQLPLUS con il comando quit), e consentirà di utilizzarlo ogni volta che si desidera. Per suggerimenti sulle modalità per automatizzare questa operazione, fare riferimento alla discussione del paragrafo “Uso di login.sql per definire l’editor”, più avanti nel capitolo.

## Il comando host

Nell’eventualità improbabile che nessuno dei comandi di editing descritti nel paragrafo precedente funzioni, se si desidera assolutamente utilizzare l’editor prescelto è possibile richiamarlo digitando:

```
host vi fred.sql
```

host indica a SQLPLUS che si tratta di un comando inteso semplicemente per restituire al sistema operativo l’esecuzione e che questo comando equivale a digitare vi fred.sql al prompt >. Per inciso, questo stesso comando host può essere utilizzato per eseguire pressoché tutti i comandi del sistema operativo da SQLPLUS, compresi dir, copy, move, erase, cls e altri ancora.

## Come aggiungere comandi SQLPLUS

Dopo avere salvato un’istruzione SQL in un file, come fred.sql, a questo file si potrà aggiungere qualsiasi comando SQLPLUS si desideri. In pratica si può procedere come per la creazione del file activity.sql della Figura 6.3. Una volta terminato di lavorare sul file, si può uscire dall’editor e tornare a SQLPLUS.

### **start**

Una volta tornati in SQLPLUS, si potranno verificare le modifiche apportate eseguendo il file appena modificato:

```
start fred.sql
```

Tutti i comandi SQLPLUS e SQL presenti nel file vengono eseguiti, linea dopo linea, proprio come se ognuno fosse stato inserito manualmente. Se nel file sono stati inseriti i comandi spool e spool off, è possibile utilizzare l’editor per visualizzare i risultati del proprio lavoro. Si tratta esattamente di ciò che era visualizzato nella Figura 6.2, il prodotto dell’avvio di activity.sql e dello spooling dei suoi risultati in activity.lst.

Per sviluppare un report, occorre eseguire ciclicamente le fasi seguenti:

1. Utilizzare SQLPLUS per creare in modo interattivo una query SQL. Quando il risultato è vicino alle aspettative, salvare la query con un nome come test.sql (l’estensione .sql viene generalmente utilizzata per i file di avvio, gli script che devono essere eseguiti per produrre un report).
2. Modificare il file test.sql utilizzando l’editor prescelto. Aggiungere al file i comandi column, break, compute, set e spool. In genere si effettua lo spooling in un file con l’estensione .lst, come test.lst. Uscire dall’editor.
3. Tornati in SQLPLUS, avviare il file test.sql con il comando start. I risultati vengono visualizzati sullo schermo e riportati nel file test.lst. Questo file viene esaminato dall’editor.
4. Incorporare qualsiasi modifica necessaria nel file test.sql ed eseguirlo nuovamente.
5. Proseguire con questa procedura finché il report non è chiaro e corretto.

## 6.3 Controllo dell’ambiente SQLPLUS

In precedenza si è segnalato che l’editor a linea di comando non è in grado di modificare i comandi SQLPLUS, in quanto agisce soltanto sulle istruzioni SQL, ovvero le linee memorizzate nel buffer SQL. Si è anche visto che esiste la possibilità di salvare le istruzioni SQL e di memorizzare le impostazioni ambientali in file nel quale possono essere modificate utilizzando il proprio editor.

Se si desidera controllare come è stata definita una determinata colonna, occorre digitare:

```
column GiorniFuori
```

senza altri elementi dopo il nome di colonna. SQLPLUS presenterà un elenco delle istruzioni impartite per quella colonna, come mostrato di seguito:

```
COLUMN GiorniFuori ON
HEADING 'Giorni!Fuori' headsep '!'
FORMAT 999.99
```

Se si digita soltanto la parola column, senza specificare il nome della colonna, verranno elencate *tutte* le colonne:

```
COLUMN Titolo ON
FORMAT a20
word_wrap

COLUMN GiorniFuori ON
HEADING 'Giorni!Fuori' headsep '!'
FORMAT 999.99

COLUMN Nome ON
FORMAT a20
```

Le istruzioni impostate con i comandi ttitle, btitle, break e compute vengono visualizzate semplicemente digitando i nomi dei comandi, senza altri elementi. SQLPLUS risponde immediatamente con le definizioni correnti. La prima linea in ciascuno degli esempi successivi è la richiesta impostata dall'utente; le linee seguenti sono le risposte di SQLPLUS:

```
Ttitle
ttitle ON and is the following 31 characters:
Registro prestiti dal 1/1/02 al 31/3/02

btitle
btitle ON and is the following 18 characters:
from the Libreria

break
break on report nodup
    on Nome skip 1 nodup

compute
COMPUTE avg LABEL 'avg' OF GiorniFuori ON Nome
COMPUTE avg LABEL 'avg' OF GiorniFuori ON report
```

Per visualizzare le impostazioni ( dette anche *parametri*) che seguono il comando set occorre utilizzare show:

```
show headsep
headsep "!" (hex 21)

show linesize
linesize 80

show pagesize
pagesize 60

show newpage
newpage 0
```

Per un elenco completo dei parametri, si rimanda al Capitolo 42.

Le impostazioni ttitle e btitle possono essere disabilitate con i comandi btitle off e ttitle off. Il listato seguente mostra questi comandi. A questi comandi non segue nessuna risposta da parte di SQLPLUS.

---

```
ttitle off
```

```
btitle off
```

Le impostazioni riferite a colonne, interruzioni e calcoli possono essere disattivate con i comandi `clear columns`, `clear breaks` e `clear computes`. La prima linea di ciascun esempio nei listati seguenti mostra l'istruzione dell'utente; le linee seguenti illustrano la risposta di SQLPLUS:

```
clear columns
columns cleared
```

```
clear breaks
breaks cleared
```

```
clear computes
computes cleared
```

## 6.4 I fondamenti

Questo capitolo è molto denso, soprattutto per i gli utenti che si avvicinano solo ora a SQLPLUS; riflettendoci però i lettori concorderanno probabilmente sul fatto che i concetti presentati non sono veramente difficili. Se la Figura 6.3 poteva apparire complessa quando si è iniziata la lettura del capitolo, ora appare sicuramente diversa. Probabilmente, tutte le linee risultano comprensibili e si riesce almeno ad avere un'idea di ciò che avviene e delle motivazioni. Volendo, è possibile copiare il file `activity.sql` in un altro file con un nome diverso e iniziare a modificarlo in modo che risponda ai propri gusti, adattandolo alle tabelle impostate. La struttura di qualsiasi report prodotto sarà in definitiva molto simile.

Gli elementi nel file `activity.sql` sono molti, tuttavia esso è composto da unità di costruzione semplici. Questo è l'approccio utilizzato per tutto il testo. Oracle fornisce unità di costruzione in gran quantità, tuttavia ciascuna di esse considerata singolarmente risulta comprensibile e rivela la propria utilità.

Nei capitoli precedenti è stato illustrato come selezionare i dati dal database, scegliendo determinate colonne e ignorandone altre e scegliendo determinate righe sulla base dei vincoli logici impostati, nonché come combinare due tabelle per ottenere informazioni non disponibili con una sola di esse.

In questo capitolo si è visto come impartire ordini che SQLPLUS possa seguire nella formattazione e produzione delle pagine e delle intestazioni di report chiaramente comprensibili.

Nei prossimi capitoli, si imparerà a modificare e formattare i dati riga per riga. Il livello di conoscenza e familiarità dovrebbe crescere capitolo dopo capitolo; al termine della Parte seconda di questo libro, l'utente dovrebbe essere in grado di produrre velocemente report molto sofisticati, con grande vantaggio per sé e per l'azienda per cui lavora.



## Capitolo 7

# Ottenere e modificare informazioni di testo

- 7.1 **Tipi di dati**
- 7.2 **Che cos'è una stringa?**
- 7.3 **Notazione**
- 7.4 **Concatenazione (||)**
- 7.5 **Come tagliare e incollare le stringhe**
- 7.6 **Le funzioni di stringa order by e where**
- 7.7 **Conclusioni**

In questo capitolo vengono introdotte le *funzioni di stringa*, strumenti software che consentono di manipolare una stringa di lettere o di altri caratteri. Per un riferimento immediato alle singole funzioni, si consulti il Capitolo 42, dove le funzioni sono elencate per nome. Questo capitolo si concentra sulla manipolazione delle stringhe di testo; per eseguire ricerche di parole (comprese quelle provenienti da radici comuni e le corrispondenze imperfette), si dovrebbe utilizzare Oracle Text, come descritto nel Capitolo 24.

In Oracle le funzioni operano in due modi. Alcune creano oggetti nuovi partendo da quelli vecchi; queste producono un risultato che è una modifica dell'informazione originale, come la trasformazione di caratteri minuscoli in maiuscoli. Altre funzioni invece producono un risultato che fornisce indicazioni sulle informazioni, come la quantità di caratteri contenuti in una parola o in una frase.

**NOTA** Se si usa PL/SQL, è possibile creare delle funzioni personalizzate con l'istruzione `create function`. Per ulteriori dettagli, fare riferimento alla Parte terza.

## 7.1 Tipi di dati

Così come le persone possono essere classificate in tipologie differenti sulla base di alcune caratteristiche (timidezza, estroversione, vivacità, stupidità e così via), anche i tipi di dati diversi possono essere classificati in *tipi di dati* sulla base di determinate caratteristiche.

Tra i tipi di dati di Oracle troviamo NUMBER, CHAR (abbreviazione di CHARACTER), DATE, VARCHAR2, LONG, RAW, LONG RAW, BLOB, CLOB e BFILE. Probabilmente i primi sono ovvi, mentre gli altri sono tipi di dati particolari e faranno la loro comparsa più avanti. Una spiegazione completa di ciascuno di questi tipi di dati è disponibile nel Capitolo 42. Nei prossimi capitoli, come nel quarto, verranno analizzati in dettaglio i vari tipi di dati. Come accade con le persone, alcuni "tipi" sono molto comuni, mentre altri sono piuttosto rari.

Se le informazioni sono di tipo carattere (VARCHAR2 o CHAR), un miscuglio di lettere, segni di interpunkzione, numeri e spazi (detti anche caratteri *alfanumerici*), occorre utilizzare funzioni di stringa per modificarle o per ottenere indicazioni su di esse. Il linguaggio SQL di Oracle mette a disposizione molti di questi strumenti.

## 7.2 Che cos'è una stringa?

Una *stringa* è un concetto semplice: un insieme di elementi allineati, come una fila di case, popcorn o perle, numeri o caratteri in una frase.

Le stringhe compaiono di frequente nella gestione delle informazioni. I nomi sono stringhe di caratteri, come “Juan L'Heureaux”. I numeri telefonici sono invece stringhe di numeri, trattini e qualche volta parentesi, come in (415) 555-2676. Persino un numero puro, come 5443702, può essere considerato come un numero oppure come una stringa di caratteri.

**NOTA** *I tipi di dati rappresentati da numeri puri (con un punto decimale e un segno meno, se necessario) prendono il nome di “NUMBER” e generalmente non vengono considerati come stringhe. Un numero può essere utilizzato per alcune operazioni che non sono possibili con una stringa e viceversa.*

Le stringhe che possono comprendere qualsiasi mescolanza di lettere, numeri, spazi e altri simboli (come i segni di interpunkzione e i caratteri speciali) prendono il nome di *stringhe di caratteri*, o semplicemente *caratteri* per brevità.

In Oracle sono disponibili due tipi di dati di stringa. Le stringhe di tipo CHAR hanno sempre una lunghezza prefissata. Se si imposta un valore di stringa con lunghezza inferiore a quella di una colonna di tipo CHAR, Oracle inserisce automaticamente nella stringa degli spazi vuoti. Quando vengono confrontate stringhe di tipo CHAR in Oracle, queste vengono considerate della stessa lunghezza con l'inserimento di spazi vuoti. Ciò significa che, se si mettono a confronto le stringhe “carattere” e “carattere” nelle colonne di tipo CHAR, Oracle considererà le due stringhe uguali. Il tipo di dati VARCHAR2 è una stringa a lunghezza variabile. Il tipo di dati VARCHAR è sinonimo di VARCHAR2, tuttavia questa equivalenza potrebbe cambiare nelle prossime versioni di Oracle, quindi si dovrebbe evitare di usare VARCHAR. CHAR si utilizza per campi con stringhe di caratteri a lunghezza prefissata, mentre VARCHAR2 ricorre in tutti gli altri campi con stringhe di caratteri.

Le semplici funzioni di stringa di Oracle spiegate in questo capitolo sono elencate nella Tabella 7.1.

## 7.3 Notazione

Le funzioni sono indicate con questo tipo di notazione:

`FUNZIONE(stringa [,opzione])`

Il nome della funzione è scritto in caratteri maiuscoli. Gli elementi cui è riferito (in genere una stringa) sono visualizzati in corsivo minuscolo. Ogni volta che compare la parola *stringa*, essa rappresenta effettivamente una stringa di caratteri oppure il nome di una colonna di caratteri in una tabella. Quando si utilizza una vera funzione di stringa, qualsiasi stringa di caratteri dovrà essere racchiusa tra apici, mentre un nome di colonna non dovrà avere nessun apice.

In tutte le funzioni compare soltanto una coppia di parentesi. Il valore su cui opera questa funzione, così come qualsiasi informazione aggiuntiva si possa passare alla funzione, dovrà essere racchiuso tra parentesi.

Alcune funzioni presentano delle *opzioni*, elementi che non sono sempre necessari perché la funzione operi come desiderato. Le opzioni sono sempre visualizzate tra parentesi quadre [ ]. Per un esempio di come vengono impiegate le opzioni, fare riferimento alla discussione su LPAD e RPAD del prossimo paragrafo.

Di seguito, viene illustrato un semplice esempio del formato della funzione LOWER:

`LOWER(stringa)`

La parola “LOWER” seguita dall’espressione tra parentesi è la funzione stessa, quindi è riportata in caratteri maiuscoli. *stringa* rappresenta invece l’effettiva stringa di caratteri che dev’essere convertita in caratteri minuscoli e viene visualizzata in corsivo minuscolo. Il comando seguente:

`LOWER('CAMP DOUGLAS')`

avrebbe come risultato:

camp douglas

La stringa ‘CAMP DOUGLAS’ è un letterale (argomento di cui si è discusso nel Capitolo 3), ovvero è letteralmente la stringa di caratteri su cui deve operare la funzione LOWER. Oracle utilizza gli apici per segnalare l’inizio e la fine di qualsiasi stringa letterale. Questa stringa nella

**Tabella 7.1** Semplici funzioni stringa di Oracle.

NOME DI FUNZIONE	USO
<code>  </code>	Unisce o concatena due stringhe. Il simbolo <code> </code> è detto barra verticale spezzata.
<code>ASCII</code>	Fornisce la rappresentazione decimale del primo carattere della stringa nel set di caratteri del database.
<code>CHR</code>	Restituisce il carattere avente l’equivalente binario della stringa nel set di caratteri del database o nel set di caratteri nazionale.
<code>CONCAT</code>	Concatena due stringhe (come <code>  </code> ).
<code>INITCAP</code>	Pone in maiuscolo la prima lettera di una parola o di una serie di parole.
<code>INSTR</code>	Trova la posizione di un carattere in una stringa.
<code>LENGTH</code>	Indica la lunghezza di una stringa.
<code>LOWER</code>	Converte ogni lettera di una stringa in minuscolo.
<code>LPAD</code>	Riempie una stringa fino a una certa lunghezza aggiungendo a sinistra una determinata serie di caratteri.
<code>LTRIM</code>	Elimina tutti i caratteri che rientrano in una serie specificata dalla parte sinistra di una stringa.
<code>RPAD</code>	Riempie una stringa fino a una certa lunghezza aggiungendo a destra una determinata serie di caratteri.
<code>RTRIM</code>	Elimina tutti i caratteri che rientrano in una serie specificata dalla parte destra di una stringa.
<code>SOUNDEX</code>	Trova parole che si pronunciano in modo simile a quelle specificate (in inglese).
<code>SUBSTR</code>	Ritaglia una porzione di una stringa.
<code>TRIM</code>	Elimina tutti i caratteri che rientrano in una serie specificata da entrambe le estremità di una stringa.
<code>UPPER</code>	Converte ogni lettera di una stringa in maiuscolo.

funzione LOWER avrebbe anche potuto rappresentare il nome di una colonna di una tabella, nel qual caso la funzione sarebbe stata operativa sul contenuto della colonna, per ogni riga restituita da un'istruzione select. Per esempio, in:

```
select Citta, LOWER(Citta), LOWER('Citta') from CLIMA;
```

si otterebbe questo risultato:

CITTA	LOWER(CITTA)	LOWER('CITTA')
LIMA	lima	Citta
PARIGI	parigi	Citta
MANCHESTER	manchester	Citta
ATENE	atene	Citta
CHICAGO	chicago	Citta
SYDNEY	sydney	Citta
SPARTA	sparta	Citta

All'inizio della seconda colonna, nella funzione LOWER, la parola CITTA non è racchiusa tra apici. In questo modo si indica a Oracle che si tratta di un nome di colonna e non di un elemento letterale.

All'inizio della terza colonna, nella funzione LOWER, la parola "CITTA" è racchiusa tra apici. Ciò significa che si desidera letteralmente che la funzione LOWER operi sulla parola "CITTA" (ovvero, la stringa costituita dalle lettere C-I-T-T-A), e non sulla colonna omonima.

## 7.4 Concatenazione (||)

Questa notazione:

```
stringa || stringa
```

indica a Oracle di concatenare, o unire, due stringhe. Queste naturalmente possono essere sia nomi di colonna, sia elementi letterali. Per esempio, in:

```
select Citta||Paese from DISLOCAZIONE;
```

CITTA    PAESE
ATENEGRECIA
CHICAGOSTATI UNITI
CONAKRYGUINEA
LIMAPERU
MADRASINDIA
MANCHESTERINGHILTERRA
MOSCARUSSIA
PARIGIFRANCIA
SHENYANGCINA
ROMITALIA
TOKYOGIAPPONE
SYDNEYAUSTRALIA
SPARTAGRECIA
MADRIDSPAGNA

In questo caso, i nomi di città hanno una lunghezza variabile da 4 a 12 caratteri. I nomi dei paesi risultano attaccati sulla destra. La funzione di concatenazione opera esattamente con questa modalità: unisce colonne o stringhe senza nessuno spazio intermedio.

La lettura del risultato non è però molto agevole. Per facilitare leggermente la lettura del risultato, si potrebbero elencare le città e i Paesi separandoli con una virgola e uno spazio. A tal fine è sufficiente concatenare le colonne Citta e Paese con una stringa letterale formata da una virgola e da uno spazio, come in questo caso:

```
select Citta || ', ' || Paese from DISLOCAZIONE;
CITTA || '.' || PAESE
-----
ATENE, GRECIA
CHICAGO, STATI UNITI
CONAKRY, GUINEA
LIMA, PERU
MADRAS, INDIA
MANCHESTER, INGHILTERRA
MOSCA, RUSSIA
PARIGI, FRANCIA
SHENYANG, CINA
ROMA, ITALIA
TOKYO, GIAPPONE
SYDNEY, AUSTRALIA
SPARTA, GRECIA
MADRID, SPAGNA
```

Si osservi il titolo di colonna. Si rimanda al Capitolo 6 per un ripasso su questo argomento. Per concatenare delle stringhe, si può anche utilizzare la funzione CONCAT. La query

```
select CONCAT(Citta, Paese) from DISLOCAZIONE;
```

equivale a questa query:

```
select Citta||Paese from DISLOCAZIONE;
```

## 7.5 Come tagliare e incollare le stringhe

In questo paragrafo vengono illustrate diverse funzioni che spesso sono fonte di confusione per gli utenti: LPAD, RPAD, LTRIM, RTRIM, TRIM, LENGTH, SUBSTR e INSTR. Hanno tutte uno scopo comune: *tagliare e incollare*.

Ciascuna di queste funzioni svolge una parte della procedura. Per esempio, LENGTH consente di conoscere il numero di caratteri contenuti in una stringa. SUBSTR consente di ritagliare e utilizzare una *sottostringa*, una porzione di una stringa, che inizia e termina in determinate posizioni all'interno della stringa. Con INSTR è possibile trovare la collocazione di un gruppo di caratteri all'interno di un'altra stringa. Con LPAD e RPAD si possono facilmente concatenare spazi o altri caratteri a sinistra o a destra di una stringa. LTRIM e RTRIM tagliano i caratteri da una delle estremità delle stringhe, mentre TRIM riesce a tagliare i caratteri da entrambe le estremità contemporaneamente. La cosa più interessante è che tutte queste funzioni possono essere utilizzate in combinazione l'una con l'altra, come verrà mostrato tra breve.

## RPAD e LPAD

RPAD e LPAD sono funzioni molto simili. La prima consente di “riempire” il lato destro di una colonna con qualsiasi gruppo di caratteri. È possibile impostare caratteri di vario tipo: spazi, punti, virgole, lettere o numeri, simboli di valuta e anche punti esclamativi (!). LPAD ha la stessa funzione di RPAD, ma agisce sul lato sinistro della colonna.

La sintassi per le funzioni RPAD e LPAD è la seguente:

```
RPAD(stringa, lunghezza [, 'car'])
```

```
LPAD(stringa, lunghezza [, 'car'])
```

*stringa* è il nome di una colonna di tipo CHAR o VARCHAR2 del database (o una stringa letterale), *lunghezza* è il numero totale di caratteri del risultato (in altre parole, la larghezza della colonna), mentre *car* è l’insieme di caratteri utilizzati per il riempimento. Il gruppo di caratteri dev’essere racchiuso tra apici . Con le parentesi quadre si segnala che il gruppo di caratteri (e la virgola che lo precede) sono facoltativi. Se non viene impostato questo parametro, il riempimento avviene automaticamente con il ricorso a degli spazi. Talvolta questa opzione prende il nome di opzione *predefinita*; ovvero, se non si indica alla funzione quale gruppo di caratteri utilizzare, questa per default utilizzerà gli spazi.

Molti utenti creano tabelle con puntini come aiuto visivo per collegare un lato della pagina all’altro. Ecco come ottenere questa impostazione con la funzione RPAD:

```
select RPAD(Città,35,'.'), Temperatura from CLIMA;
```

	TEMPERATURA
LIMA.....	7
PARIGI.....	27
MANCHESTER.....	19
ATENE.....	36
CHICAGO.....	19
SYDNEY.....	-5
SPARTA.....	74

Si osservi che cosa è accaduto. RPAD ha preso ogni città, da Lima a Sparta, ha concatenato dei puntini alla destra di ogni città, aggiungendone per ognuna un numero sufficiente a fare in modo che il risultato (città più puntini) fosse lungo esattamente 35 caratteri. La funzione di concatenamento ( || ) non avrebbe potuto ottenere questo risultato. Avrebbe aggiunto lo stesso numero di puntini per ogni città, lasciando sulla destra un bordo irregolare.

Con LPAD si ottiene lo stesso risultato, ma sul lato sinistro. Si supponga di voler formattare nuovamente le città e le temperature in modo che i nomi di città siano giustificati a destra (ovvero, allineati a destra):

```
select LPAD(Città,11), Temperatura from CLIMA;
```

	TEMPERATURA
LIMA	7
PARIGI	27
MANCHESTER	19
ATENE	36
CHICAGO	19

SYDNEY	-5
SPARTA	23

## LTRIM, RTRIM e TRIM

LTRIM e RTRIM sono come dei potasiepi, in quanto tagliano i caratteri indesiderati dall'estremità destra e sinistra delle stringhe. Per esempio, si supponga di avere una tabella RIVISTA con un colonna che contiene i titoli degli articoli, e che questi titoli siano stati inseriti da persone diverse. Alcuni hanno inserito i titoli utilizzando sempre le virgolette, altri hanno semplicemente digitato il titolo; alcuni hanno utilizzato dei punti, altri no; alcuni hanno posto l'articolo davanti ai titoli, altre no. Com'è possibile tagliare questi elementi?

```
select Titolo from RIVISTA;
```

TITOLO

```
-----  
THE BARBERS WHO SHAVE THEMSELVES.  
"HUNTING THOREAU IN NEW HAMPSHIRE"  
THE ETHNIC NEIGHBORHOOD  
RELATIONAL DESIGN AND ENTHALPY  
"INTERCONTINENTAL RELATIONS."
```

La sintassi per le funzioni RTRIM e LTRIM è la seguente:

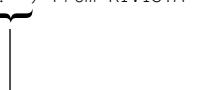
```
RTRIM(stringa [, 'car'])
```

```
LTRIM(stringa [, 'car'])
```

*stringa* è il nome della colonna del database (o una stringa letterale), e *set* è la collezione dei caratteri che si desidera eliminare. Se non viene specificato nessun set di caratteri, vengono rimossi gli spazi.

Si può eliminare più di un carattere per volta; a questo scopo, occorre semplicemente creare un elenco (una stringa) dei caratteri che si desidera rimuovere. Innanzitutto, si eliminano le virgolette e i punti sulla destra, come mostrato di seguito:

```
select RTRIM(Titolo,'.')
```



Insieme di caratteri da eliminare

Il codice precedente produce quanto segue:

```
RTRIM(TITOLO,'.')
```

```
-----  
THE BARBERS WHO SHAVE THEMSELVES  
"HUNTING THOREAU IN NEW HAMPSHIRE"  
THE ETHNIC NEIGHBORHOOD  
RELATIONAL DESIGN AND ENTHALPY  
"INTERCONTINENTAL RELATIONS"
```

Con RTRIM sono stati rimossi sia i punti sia le virgolette dalla destra dei titoli. Il gruppo di caratteri che si intende rimuovere può avere una lunghezza desiderata. Viene controllato e

ricontrollato il lato destro di ogni titolo finché non è stato rimosso ogni carattere della stringa, ovvero finché non si incontra il primo carattere della stringa che *non* era stato indicato nel gruppo di caratteri da eliminare.

## Combinazione di due funzioni

E ora? Come sbarazzarsi delle virgolette sulla sinistra? Esistono due opzioni, la prima delle quali ricorre alla funzione LTRIM. In questo paragrafo, si imparerà a combinare delle funzioni.

Come si è già detto, quando è stata eseguita l'istruzione select:

```
select Titolo from RIVISTA;
```

si è ottenuto come risultato il contenuto della colonna Titolo, come mostrato di seguito:

```
THE BARBERS WHO SHAVE THEMSELVES.  
"HUNTING THOREAU IN NEW HAMPSHIRE"  
THE ETHNIC NEIGHBORHOOD  
RELATIONAL DESIGN AND ENTHALPY  
"INTERCONTINENTAL RELATIONS."
```

Si ricordi che lo scopo di:

```
RTRIM(Titolo, '...')
```

è prendere ciascuna di queste stringhe e rimuovere le virgolette dal lato destro, producendo in realtà un risultato che equivale a una *nuova* colonna il cui contenuto è visualizzato di seguito:

```
THE BARBERS WHO SHAVE THEMSELVES  
"HUNTING THOREAU IN NEW HAMPSHIRE  
THE ETHNIC NEIGHBORHOOD  
RELATIONAL DESIGN AND ENTHALPY  
"INTERCONTINENTAL RELATIONS
```

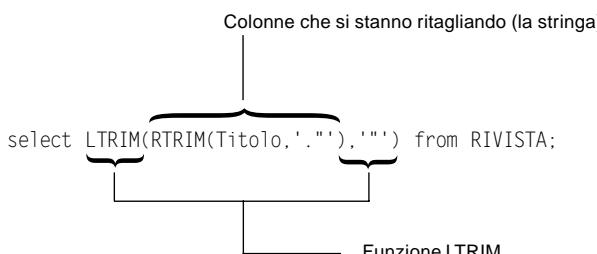
Pertanto, se si suppone che RTRIM(Titolo,'...') sia semplicemente un nome di colonna, è possibile sostituirlo al termine *stringa* in:

```
LTRIM(stringa, 'car')
```

Quindi è sufficiente digitare l'istruzione select in questo modo:

```
select LTRIM(RTRIM(Titolo, '...'), '...') from RIVISTA;
```

Seguendo questo schema, l'istruzione risulterà più chiara:



È questa la modalità da utilizzare? E qual è il risultato di questa funzione combinata?

```
LTRIM(RTRIM(TITOLO,'.'),'''')
```

```
-----  
THE BARBERS WHO SHAVE THEMSELVES  
HUNTING THOREAU IN NEW HAMPSHIRE  
THE ETHNIC NEIGHBORHOOD  
RELATIONAL DESIGN AND ENTHALPY  
INTERCONTINENTAL RELATIONS
```

I titoli sono stati completamente ripuliti.

In effetti osservare una combinazione di funzioni per la prima volta (o anche per la centesima) può generare confusione, anche per un utente esperto di query. È difficile stabilire quali virgolette e quali parentesi fanno parte della funzione, soprattutto quando una query impostata non funziona correttamente; scoprire dove manca una virgola o quale parentesi non corrisponde correttamente a un'altra può essere davvero difficile.

Una soluzione semplice consiste nel suddividere le funzioni in linee separate, almeno finché non funzionano tutte nella maniera desiderata. Per SQLPLUS non ha nessuna importanza il punto in cui viene interrotta un'istruzione SQL, purché non sia nel mezzo di una parola o di una stringa letterale. Per far comprendere meglio come funziona questa combinazione di RTRIM e LTRIM, si potrebbe digitarla in questo modo:

```
select LTRIM(  
    RTRIM(Titolo,'.'),  
    ,''')  
from RIVISTA;
```

Così lo scopo di quello che si cerca di fare risulta ovvio, e il tutto funzionerà ugualmente anche se digitato su quattro linee separate e con molti spazi. SQLPLUS ignora gli spazi in più.

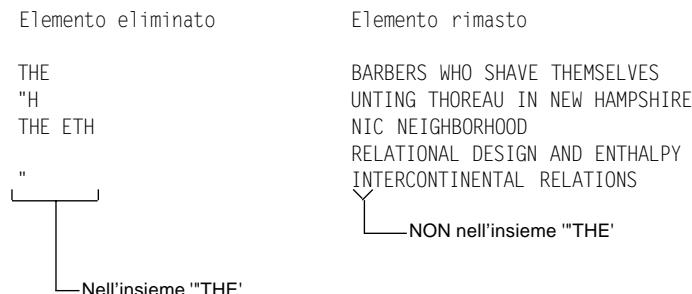
Ora, si supponga di voler eliminare l'articolo “THE” dall'inizio dei due titoli, con lo spazio che lo segue (e naturalmente le virgolette che sono state rimosse in precedenza). Si potrebbe procedere così:

```
select LTRIM(RTRIM(Titolo,'.'),'"THE ')  
from RIVISTA;
```

con questo risultato:

```
LTRIM(RTRIM(TITOLO,'.'),'"I ')  
-----  
BARBERS WHO SHAVE THEMSELVES  
UNTING THOREAU IN NEW HAMPSHIRE  
NIC NEIGHBORHOOD  
RELATIONAL DESIGN AND ENTHALPY  
INTERCONTINENTAL RELATIONS
```

Che cosa è accaduto? La seconda e la terza riga sono state troncate più di quanto desiderato. Perché? Perché la funzione LTRIM era impegnata a cercare ed eliminare qualsiasi cosa fosse una virgoletta, *T*, *H*, *E* o spazio. Non stava cercando l'articolo “THE”. Bensì, le lettere in esso contenute, dimenticandosi di terminare l'operazione la prima volta che ha incontrato una delle lettere che stava cercando. È infatti terminata nel momento in cui ha visto un carattere che non era compreso nel gruppo.



In altre parole, con tutte le combinazioni seguenti:

- ' "THE'
- 'HET" '
- 'E"TH'
- 'H"TE'
- 'ET"H'

e con molte altre combinazioni di queste lettere si ottiene lo stesso effetto, quando vengono impostate come gruppo di un comando LTRIM o RTRIM. L'ordine delle lettere del gruppo non ha nessun effetto sulle modalità operative della funzione. Tuttavia, si noti che il *carattere maiuscolo o minuscolo* delle lettere è importante. Oracle controllerà il carattere maiuscolo o minuscolo delle lettere sia nel gruppo sia nella stringa. Vengono rimosse soltanto le lettere esattamente corrispondenti.

LTRIM e RTRIM sono progettate per rimuovere qualsiasi carattere contenuto in un gruppo partendo dal lato sinistro o destro di una stringa: il loro obiettivo non è rimuovere le parole. Per ottenere questo risultato occorre utilizzare in maniera accorta le funzioni INSTR, SUBSTR e anche DECODE, illustrate nel Capitolo 16.

L'esempio precedente chiarisce un punto: è meglio accertarsi che i dati vengano ripuliti o modificati prima di essere memorizzati nel database. Vi sarebbero stati molti meno problemi se le persone che hanno inserito i titoli degli articoli della rivista avessero semplicemente evitato l'utilizzo di virgolette, punti e della parola "THE".

## Uso della funzione TRIM

Nell'esempio precedente si è spiegato come combinare due funzioni, una soluzione molto utile quando si ha la necessità di manipolare delle stringhe. Se si devono eliminare esattamente gli stessi dati da entrambe le estremità della stringa, è possibile ricorrere alla funzione TRIM al posto di una combinazione di LTRIM/RTRIM.

TRIM utilizza una sintassi unica. L'esempio seguente illustra l'impiego della funzione TRIM e della clausola ad essa associata, from. In questo esempio, le virgolette vengono rimosse dall'inizio e dalla fine dei titoli degli articoli contenuti nella rivista. Dato che una virgola è una stringa di caratteri, viene racchiusa tra due apici:

```
select TRIM(from Titolo) from RIVISTA;
```

```
TRIM('' FROMTITOLO)
```

```
-----  
THE BARBERS WHO SHAVE THEMSELVES.  
HUNTING THOREAU IN NEW HAMPSHIRE
```

THE ETHNIC NEIGHBORHOOD  
 RELATIONAL DESIGN AND ENTHALPY  
 INTERCONTINENTAL RELATIONS.

Le virgolette sono state rimosse dall'estremità iniziale e da quella finale delle stringhe. Se si desidera tagliare soltanto un'estremità delle stringhe, si potrebbero utilizzare le clausole leading o trailing, come mostrato nel listato seguente.

```
select TRIM(leading '' from Titolo) from RIVISTA;
select TRIM(trailing '' from Titolo) from RIVISTA;
```

L'uso di leading fa sì che TRIM si comporti come LTRIM, mentre con trailing si comporta come RTRIM. L'aspetto più utile di TRIM è la sua capacità di operare contemporaneamente su entrambe le estremità di una stringa, semplificando il codice da scrivere, in quanto dalle due estremità della stringa verranno rimossi gli stessi caratteri.

## Aggiunta di un'altra funzione

Si supponga di voler riempire il titolo ripulito con trattini e accenti circonflessi utilizzando la funzione RPAD, magari selezionando il nome e il numero di pagina di una rivista. La query avrebbe questo aspetto:

```
select Nome, RPAD(RTRIM(LTRIM(Titolo,'')),'.',''),47,'-^'), Pagina
      from RIVISTA;
Nome          RPAD(RTRIM(LTRIM(TITOLO,'')),'.',''),47,'-^')  PAGINA
----- -----
BERTRAND MONTHLY THE BARBERS WHO SHAVE THEMSELVES-^-^--^-^--^  70
LIVE FREE OR DIE HUNTING THOREAU IN NEW HAMPSHIRE-^-^--^-^--^  320
PSYCHOLOGICA    THE ETHNIC NEIGHBORHOOD-^-^--^-^--^-^--^  246
FADED ISSUES    RELATIONAL DESIGN AND ENTHALPY-^-^--^-^--^--^  279
ENTROPY WIT     INTERCONTINENTAL RELATIONS-^-^--^-^--^-^--^  20
```

In ogni funzione sono presenti delle parentesi che racchiudono la colonna su cui opera la funzione stessa, quindi il vero trucco per comprendere le funzioni combinate nelle istruzioni select consiste nel leggere dall'esterno verso l'interno sia a sinistra sia a destra, osservando (e anche contando) le coppie di parentesi.

## LOWER, UPPER e INITCAP

Queste tre semplicissime funzioni correlate vengono spesso utilizzate insieme. Con la funzione LOWER le lettere di qualsiasi stringa o colonna vengono convertite in minuscole. Con la funzione UPPER si ottiene il risultato opposto, ossia la trasformazione di qualsiasi lettera in maiuscolo. Con la funzione INITCAP la lettera iniziale di ogni parola in una stringa o in una colonna viene convertita in maiuscolo.

La sintassi per queste tre funzioni è la seguente:

```
LOWER(stringa)
UPPER(stringa)
INITCAP(stringa)
```

Ritornando alla tabella CLIMA, in cui ogni città era memorizzata in lettere maiuscole:

LIMA  
PARIGI  
ATENE  
CHICAGO  
MANCHESTER  
SYDNEY  
SPARTA

il comando seguente:

```
select Citta, UPPER(Citta), LOWER(Citta), INITCAP(LOWER(Citta))
  from CLIMA;
```

produce:

Citta	UPPER(CITTA)	LOWER(CITTA)	INITCAP(LOW
LIMA	LIMA	lima	Lima
PARIGI	PARIGI	parigi	Parigi
MANCHESTER	MANCHESTER	manchester	Manchester
ATENE	ATENE	atene	Atene
CHICAGO	CHICAGO	chicago	Chicago
SYDNEY	SYDNEY	sydney	Sydney
SPARTA	SPARTA	sparta	Sparta

Si osservino con attenzione il risultato prodotto in ciascuna colonna e le funzioni che hanno prodotto questo risultato nell'istruzione SQL. Nella quarta colonna è visualizzata la modalità di applicazione della funzione INITCAP a LOWER(Citta): i nomi delle città compaiono nella forma normale, anche se sono memorizzati in lettere maiuscole.

Un altro esempio è costituito dalla colonna Nome come appare memorizzata nella tabella RIVISTA:

NOME
-----
BERTRAND MONTHLY
LIVE FREE OR DIE
PSYCHOLOGICA
FADED ISSUES
ENTROPY WIT

che viene richiamata con la combinazione delle funzioni INITCAP e LOWER, come mostrato di seguito:

```
select INITCAP(LOWER(Nome)) from RIVISTA;
```

INITCAP(LOWER(NO
-----
Bertrand Monthly
Live Free Or Die
Psychologica
Faded Issues
Entropy Wit

Ecco come appare dopo l'applicazione delle stesse funzioni a Nome, al titolo ripulito e al numero di pagina (si osservi che vengono anche rinominate le colonne):

```
select INITCAP(LOWER(Nome)) Nome,
       INITCAP(LOWER(RTRIM(LTRIM(Titolo,''), '.'))) Titolo,
       Pagina
  from RIVISTA;
```

NOME	TITOLO	PAGINA
Bertrand Monthly	The Barbers Who Shave Themselves	70
Live Free Or Die	Hunting Thoreau In New Hampshire	320
Psychologica	The Ethnic Neighborhood	246
Faded Issues	Relational Design And Enthalpy	279
Entropy Wit	Intercontinental Relations	20

## LENGTH

Si tratta di una funzione di facile utilizzo. Con LENGTH viene segnalata la lunghezza di una stringa, ovvero il numero di caratteri che la costituiscono, compresi lettere, spazi e qualsiasi altro elemento.

La sintassi per la funzione LENGTH è la seguente:

LENGTH(*stringa*)

Per esempio, in:

```
select Nome, LENGTH(Nome) from RIVISTA;
```

NOME	LENGTH(NOME)
BERTRAND MONTHLY	16
LIVE FREE OR DIE	16
PSYCHOLOGICA	12
FADED ISSUES	12
ENTROPY WIT	11

Di per sé, non è normalmente una funzione molto utile, tuttavia può essere utilizzata come parte di un'altra funzione, per il calcolo dello spazio necessario in un report, o come parte di clausole come where o order by.

**NOTA** *Non è possibile eseguire funzioni come LENGTH su una colonna che utilizza un tipo di dato LONG.*

## SUBSTR

È possibile utilizzare la funzione SUBSTR per estrarre una parte di una stringa.

La sintassi per SUBSTR è la seguente:

SUBSTR(*stringa*,*inizio* [*.conta*])

Con questa notazione si indica a SQL di estrarre una sottosezione della *stringa*, iniziando dalla posizione segnalata da *inizio* e proseguendo per *conta* caratteri. Se non viene specificata la posizione *conta*, SUBSTR inizierà l'operazione partendo da *inizio* e continuando fino alla fine della stringa. Per esempio, nell'istruzione:

```
select SUBSTR(Nome,6,4) from RIVISTA;
```

si ottiene:

```
SUBS  
---  
AND  
FREE  
OLOG  
ISS  
PY W
```

Il modo di operare della funzione è abbastanza chiaro. Sono state ritagliate parti del nome della rivista iniziando dalla posizione 6 (contando da sinistra) e comprendendo un totale di quattro caratteri.

Un impiego più pratico di questa funzione potrebbe essere la selezione di numeri telefonici da una rubrica personale. Per esempio, si supponga di avere una tabella INDIRIZZO in cui siano contenuti, tra le altre cose, cognomi, nomi e numeri di telefono, come nel caso seguente:

```
select Cognome, Nome, Telefono from INDIRIZZO;
```

COGNOME	NOME	TELEFONO
BAILEY	WILLIAM	213-293-0223
ADAMS	JACK	415-453-7530
SEP	FELICIA	214-522-8383
DE MEDICI	LEFTY	312-736-1166
DEMIURGE	FRANK	707-767-8900
CASEY	WILLIS	312-684-1414
ZACK	JACK	415-620-6842
YARROW	MARY	415-787-2178
WERSCHKY	ARNY	415-235-7387
BRANT	GLEN	415-526-7512
EDGAR	THEODORE	415-525-6252
HARDIN	HUGGY	617-566-0125
HILD	PHIL	603-934-2242
LOEBEL	FRANK	202-456-1414
MOORE	MARY	718-857-1638
SZEP	FELICIA	214-522-8383
ZIMMERMAN	FRED	503-234-7491

Si supponga di voler estrapolare solamente i numeri di telefono con il prefisso telefonico 415. Una soluzione consisterebbe nell'avere una colonna separata di nome Prefisso. L'attenta pianificazione degli elementi di tabelle e colonne consente di eliminare molte operazioni successive di riformattazione. Tuttavia, in questo caso i prefissi e i numeri di telefono sono riuniti in un'unica colonna, quindi occorre trovare il modo per selezionare i numeri con il prefisso 415.

```
select Cognome, Nome, Telefono from INDIRIZZO  
where Telefono like '415-%';
```

COGNOME	NOME	TELEFONO
ADAMS	JACK	415-453-7530
ZACK	JACK	415-620-6842
YARROW	MARY	415-787-2178

WERSCHKY	ARNY	415-235-7387
BRANT	GLEN	415-526-7512
EDGAR	THEODORE	415-525-6252

Successivamente, poiché non è necessario digitare il prefisso quando si chiamano i numeri urbani, è possibile eliminare questo prefisso utilizzando un altro comando con la funzione SUBSTR:

```
select Cognome, Nome, SUBSTR(Telefono,5) from INDIRIZZO
where Telefono like '415-%';
```

COGNOME	NOME	SUBSTR(P)
ADAMS	JACK	453-7530
ZACK	JACK	620-6842
YARROW	MARY	787-2178
WERSCHKY	ARNY	235-7387
BRANT	GLEN	526-7512
EDGAR	THEODORE	525-6252

Si osservi che in questo caso è stata utilizzata la versione predefinita della funzione SUBSTR. SUBSTR(Telefono,5) indica a SQL di estrarre la sottocorona del numero di telefono, iniziando dalla posizione 5 e proseguendo fino alla fine della stringa. In questo modo viene eliminato il prefisso.

Naturalmente, questo comando:

```
SUBSTR(Telefono,5)
```

ha esattamente lo stesso effetto del seguente:

```
SUBSTR(Telefono,5,8)
```

È possibile utilizzare tecniche di concatenazione e ridenominazione delle colonne, come quelle trattate nel Capitolo 6, per ottenere velocemente l'elenco dei numeri telefonici urbani, come illustrato di seguito:

```
select Cognome ||', '||Nome Nome, SUBSTR(Telefono,5) Telefono from INDIRIZZO
where Telefono like '415-%';
```

NOME	TELEFONO
ADAMS, JACK	453-7530
ZACK, JACK	620-6842
YARROW, MARY	787-2178
WERSCHKY, ARNY	235-7387
BRANT, GLEN	526-7512
EDGAR, THEODORE	525-6252

Per aggiungere una linea punteggiata dopo il nome, si fa ricorso alla funzione RPAD:

```
select RPAD(Cognome ||', '||Nome,25,'.') Nome,
       SUBSTR(Telefono,5) Telefono
  from INDIRIZZO
 where Telefono like '415-%';
```

NOME	TELEFONO
ADAMS, JACK.....	453-7530
ZACK, JACK.....	620-6842
YARROW, MARY.....	787-2178
WERSCHKY, ARNY.....	235-7387
BRANT, GLEN.....	526-7512
EDGAR, THEODORE.....	525-6252

Nella funzione SUBSTR è anche possibile inserire numeri negativi. Normalmente, il valore di posizione specificato per la posizione iniziale riguarda il punto di inizio della stringa. Quando per il valore di posizione si usa un numero negativo, questo riguarda la *fine* della stringa. Per esempio, in:

SUBSTR(Teléfono, -4)

come punto iniziale verrebbe utilizzata la quarta posizione dalla fine del valore della colonna Teléfono. Poiché in questo esempio non viene specificato nessun parametro di lunghezza, si ottiene tutta la parte rimanente della stringa.

**NOTA** È opportuno utilizzare questa caratteristica soltanto per colonne con tipo di dati VARCHAR2, evitando di utilizzarla con colonne impostate con tipo di dati CHAR. Le colonne di tipo CHAR hanno lunghezza prefissata, quindi i valori sono completati con spazi in modo da occupare la lunghezza totale della colonna. Se si utilizza un numero negativo come valore di posizione SUBSTR in una colonna CHAR, la posizione di inizio viene determinata relativamente alla fine della colonna, non alla fine della riga.

Nell'esempio seguente viene illustrato il risultato di questa caratteristica applicata a una colonna VARCHAR2:

```
select SUBSTR(Teléfono, -4)
  from INDIRIZZO
 where Teléfono like '415-5%';
```

SUBS  
----  
7512  
6252

Il valore *conta* della funzione SUBSTR deve sempre essere positivo, qualora venga specificato. L'uso di un numero negativo per il valore *conta* comporterà la restituzione di un risultato NULL.

## INSTR

La funzione INSTR consente di effettuare ricerche semplici o complesse di un dato set di caratteri in una stringa, in maniera analoga alle funzioni LTRIM e RTRIM, con la differenza che con INSTR non viene eliminato alcunché. Con questa funzione infatti viene semplicemente visualizzata la posizione dell'elemento ricercato all'interno della stringa. Si tratta di una caratteristica simile all'operatore logico LIKE descritto nel Capitolo 3, con la differenza che LIKE può essere utilizzato soltanto in una clausola where o having, mentre INSTR può essere utilizzato in qualunque clausola, con l'eccezione di quelle con from. Naturalmente, LIKE può essere utilizzato per ricer-

che complesse di modelli che sarebbe molto difficile, se non impossibile, effettuare utilizzando **INSTR**.

La sintassi per la funzione **INSTR** è la seguente:

```
INSTR(stringa, set [,inizio [,occorrenza ] ])
```

Con **INSTR** viene ricercato un determinato *set* di caratteri in una *stringa*. È possibile impostare due opzioni, che si trovano una all'interno dell'altra; la prima è quella predefinita: cercherà il set di caratteri partendo dalla posizione 1. Se si specifica la posizione da cui iniziare, tutti i caratteri precedenti al punto indicato vengono saltati e la ricerca comincia esattamente da quel punto.

La seconda opzione è *occorrenza*. Un *set* di caratteri può essere ricorrente in una stringa e talvolta si è interessati proprio a tale ricorrenza. Per default, **INSTR** ricerca la prima occorrenza del set di caratteri. Aggiungendo l'opzione *occorrenza* con valore pari a 3, per esempio, si può ottenere che vengano saltate le prime due occorrenze del set di caratteri e che venga invece localizzata la terza.

Tutto ciò diventa più semplice da comprendere con l'ausilio di qualche esempio. Si riprenda la tabella con gli articoli di una rivista. Ecco un elenco degli autori:

```
select Autore from RIVISTA;
```

AUTORE

```
-----  
BONHOEFFER, DIETRICH  
CHESTERTON, G. K.  
RUTH, GEORGE HERMAN  
WHITEHEAD, ALFRED  
CROOKES, WILLIAM
```

Per trovare la posizione della prima occorrenza della lettera 'O', viene utilizzata la funzione **INSTR** senza nessuna opzione e con il *set* di caratteri indicato come 'O' (si osservi la presenza degli apici, poiché si tratta di un elemento letterale), come mostrato nel listato seguente:

```
select Autore, INSTR(Autore,'O') from RIVISTA;
```

AUTORE	INSTR(AUTORE,'O')
BONHOEFFER, DIETRICH	2
CHESTERTON, G. K.	9
RUTH, GEORGE HERMAN	9
WHITEHEAD, ALFRED	0
CROOKES, WILLIAM	3

Questo comando, naturalmente, è equivalente a quest'altro:

```
select Autore, INSTR(Autore,'O',1,1) from RIVISTA;
```

Nel caso in cui si debba ricercare la seconda occorrenza della lettera 'O' , il risultato sarebbe:

```
select Autore, INSTR(Autore,'O',1,2) from RIVISTA;
```

AUTORE	INSTR(AUTORE,'O',1,2)
BONHOEFFER, DIETRICH	5

CHESTERTON, G. K.	0
RUTH, GEORGE HERMAN	0
WHITEHEAD, ALFRED	0
CROOKES, WILLIAM	4

Con INSTR viene trovata la seconda ‘O’ nel nome Bonhoeffer, alla posizione 5, e nel nome Crookes, alla posizione 4. Nel nome Chesterton c’è una sola ‘O’ e quindi, come per Ruth, e Whitehead il risultato è zero, perché non è stata trovata una seconda ‘O’.

Per indicare a INSTR di cercare la seconda occorrenza, occorre indicare anche la posizione in cui iniziare la ricerca (in questo caso, la posizione 1). Il valore predefinito per *inizio* è 1, ovvero il valore che è utilizzato se non ne vengono specificati altri, tuttavia per impostare l’opzione *occorrenza* occorre stabilire una posizione iniziale con l’opzione *inizio*, quindi è necessario specificarle entrambe.

Se il *set* di caratteri non è composto da un solo carattere ma da molti, con INSTR viene fornita la posizione della prima lettera del gruppo, come in questo esempio:

```
select Autore, INSTR(Autore,'WILLIAM') from RIVISTA;
```

AUTORE	INSTR(AUTORE,'WILLIAM')
BONHOEFFER, DIETRICH	0
CHESTERTON, G. K.	0
RUTH, GEORGE HERMAN	0
WHITEHEAD, ALFRED	0
CROOKES, WILLIAM	10

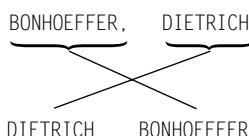
Questa possibilità può consentire diverse applicazioni utili. Nella tabella RIVISTA, per esempio:

```
select Autore, INSTR(Autore,',') from RIVISTA;
```

AUTORE	INSTR(AUTORE,',')
BONHOEFFER, DIETRICH	11
CHESTERTON, G. K.	11
RUTH, GEORGE HERMAN	5
WHITEHEAD, ALFRED	10
CROOKES, WILLIAM	8

In questo caso, con INSTR è stata impostata la ricerca di una virgola nelle stringhe dei nomi di autori, quindi è stata visualizzata la posizione in cui è stata trovata la virgola nella stringa.

Si supponga di voler formattare nuovamente i nomi degli autori in modo diverso dall’approccio formale “cognome /virgola/nome” e di volerli presentare nella forma più comunemente utilizzata, come mostrato di seguito:



Per ottenere questo risultato con INSTR e SUBSTR, occorre trovare la posizione della virgola, quindi utilizzare questa posizione per indicare a SUBSTR il punto in cui ritagliare. Seguendo la procedura passo per passo, innanzitutto si imposta la ricerca della virgola:

```
select Autore, INSTR(Autore,',') from RIVISTA;
```

AUTORE	INSTR(AUTORE,'.')
BONHOEFFER, DIETRICH	11
CHESTERTON, G. K.	11
RUTH, GEORGE HERMAN	5
WHITEHEAD, ALFRED	10
CROOKES, WILLIAM	8

È necessario utilizzare due volte la funzione SUBSTR, una per ritagliare il cognome dell'autore e collocarlo nella posizione prima della virgola, l'altra per ritagliare il nome dell'autore da due posizioni dopo la virgola e posizionarlo verso la fine della stringa.

Ecco la prima operazione con SUBSTR, in cui la posizione 1 viene ritagliata e collocata alla posizione immediatamente precedente la virgola:

```
select Autore, SUBSTR(Autore,1,INSTR(Autore,',')-1)
      from RIVISTA;
```

AUTORE	SUBSTR(AUTORE,1,INSTR(AUT
BONHOEFFER, DIETRICH	BONHOEFFER
CHESTERTON, G. K.	CHESTERTON
RUTH, GEORGE HERMAN	RUTH
WHITEHEAD, ALFRED	WHITEHEAD
CROOKES, WILLIAM	CROOKES

Successivamente si procede con la seconda, in cui l'elemento collocato nella seconda posizione dopo la virgola viene spostato alla fine della stringa:

```
select Autore, SUBSTR(Autore,INSTR(Autore,',')+2) from RIVISTA;
```

AUTORE	SUBSTR(AUTORE,INSTR(AUTO
BONHOEFFER, DIETRICH	DIETRICH
CHESTERTON, G. K.	G. K.
RUTH, GEORGE HERMAN	GEORGE HERMAN
WHITEHEAD, ALFRED	ALFRED
CROOKES, WILLIAM	WILLIAM

Si osservi la combinazione di queste due operazioni, ottenuta inserendo uno spazio tra di esse con la funzione di concatenazione, e il rapido cambio del nome della colonna in "PerNome":

```
column PerNome heading "Per nome"
```

```
select Autore, SUBSTR(Autore,INSTR(Autore,',')+2)
      ||' ' ||
      SUBSTR(Autore,1,INSTR(Autore,',')-1)
      PerNome
from RIVISTA;
```

AUTORE	Per nome
BONHOEFFER, DIETRICH	DIETRICH BONHOEFFER
CHESTERTON, G.K.	G.K. CHESTERTON
RUTH, GEORGE HERMAN	GEORGE HERMAN RUTH

WHITEHEAD, ALFRED  
CROOKES, WILLIAM

ALFRED WHITEHEAD  
WILLIAM CROOKES

Un'istruzione SQL di questo tipo può apparire oscura, tuttavia è stata creata utilizzando una logica semplice e può essere scomposta nella stessa maniera. Si può provare con Bonhoeffer.

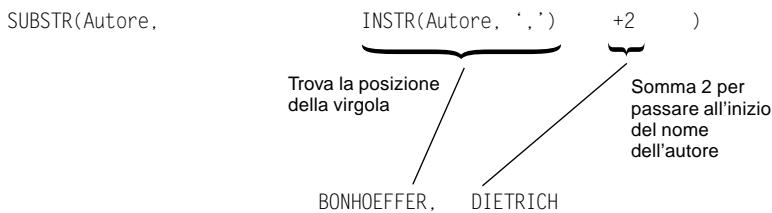
La prima parte della procedura è la seguente:

`SUBSTR(Autore,INSTR(Autore,',')+2)`

In questo modo, si indica a SQL di visualizzare la sottostringa di Autore iniziando dalla seconda posizione a destra della virgola e proseguendo fino alla fine. Così viene ritagliato "DIETRICH", il nome dell'autore.

L'inizio del nome dell'autore viene trovato localizzando la virgola alla fine del cognome (con la funzione `INSTR`) e quindi spostandosi di due posizioni sulla destra (nel punto in cui inizia il nome).

Nell'illustrazione seguente viene mostrato come opera la funzione `INSTR` (più 2) per segnalare il punto di *inizio* della funzione `SUBSTR`:



Questa è la seconda parte dell'istruzione combinata:

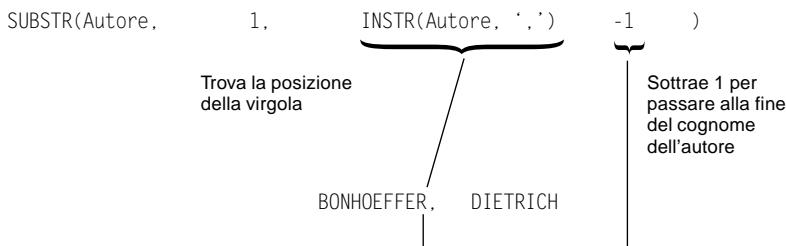
`|| ' ' ||`

che indica semplicemente a SQL di concatenare uno spazio nel mezzo.

Questa è la terza parte dell'istruzione combinata:

`SUBSTR(Autore,1,INSTR(Autore,',')-1)`

In questo modo, si indica a SQL di ritagliare la porzione del nome dell'autore iniziando dalla posizione 1 e terminando una posizione prima della virgola, ovvero dove si trova il cognome dell'autore:



La quarta parte assegna solamente un alias alla colonna:

`PerNome`

È stato possibile effettuare questa trasposizione soltanto perché, per ogni record Autore nella tabella RIVISTA, sono state seguite le stesse convenzioni di formattazione. In ciascun record, il cognome era sempre la prima parola della stringa ed era immediatamente seguito da una virgola. In questo modo, è stato possibile utilizzare la funzione INSTR per cercare la virgola. Una volta nota la posizione della virgola, è stato possibile determinare quale parte della stringa era il cognome e il resto è stato considerato come nome.

Non è sempre così. Infatti, è difficile imporre dei formati standard ai nomi. I cognomi possono avere dei prefissi (come “von” di “von Hagel”) o suffissi (come “Jr.”, “Sr.” e “III”). Se si utilizza l’istruzione SQL dell’esempio precedente, il nome “Richards, Jr., Bob” sarebbe stato trasformato in “Jr., Bob Richards”.

A causa della mancanza di un formato standard per i nomi, in molte applicazioni il nome e il cognome vengono memorizzati separatamente. Titoli (come “Dott.”) sono solitamente memorizzati in un’altra colonna. Un’altra opzione quando si inseriscono i dati di questo tipo consiste nell’applicare un unico formato e utilizzare le funzioni SUBSTR e INSTR per manipolare i dati se necessario.

## ASCII e CHR

Le funzioni ASCII e CHR sono utilizzate raramente durante le query ad hoc. CHR converte i valori numerici nei loro equivalenti stringa di caratteri ASCII:

```
select CHR(70)||CHR(83)||CHR(79)||CHR(85)||CHR(71)
      as ChrValues
     from DUAL;

CHRVA
-----
FSOUG
```

Oracle ha convertito CHR(70) in una ‘F’, CHR(83) in una ‘S’ e così via basandosi sul set di caratteri del database.

La funzione ASCII esegue l’operazione inversa; se però a questa funzione si passa una stringa, essa opererà solamente sul primo carattere della stringa:

```
select ASCII('FSOUG') from DUAL;

ASCII('FSOUG')
-----
70
```

Per visualizzare ogni valore ASCII, occorre valutare ciascuna lettera con esecuzioni separate della funzione ASCII.

## 7.6 Le funzioni di stringa con order by e where

Le funzioni di stringa possono essere utilizzate in una clausola where, come mostrato di seguito:

```
select Citta
      from CLIMA
```

```
where LENGTH(Citta) 7;
```

```
CITTA
```

```
-----  
LIMA  
PARIGI  
ATENE  
SPARTA  
SYDNEY
```

Le stesse funzioni possono essere utilizzate anche in una clausola **order by**, come mostrato di seguito:

```
select Citta  
      from CLIMA  
     order by LENGTH(Citta);
```

```
CITTA
```

```
-----  
LIMA  
PARIGI  
ATENE  
SPARTA  
SYDNEY  
CHICAGO  
MANCHESTER
```

Quelli illustrati sono esempi semplici; si potrebbero impostare clausole molto più complesse. Per esempio, si potrebbero cercare tutti gli autori i cui nomi contengono più di una ‘O’, utilizzando **INSTR** nella clausola **where**:

```
select Autore from RIVISTA  
  where INSTR(Autore,'O',1,2) > 0;
```

```
AUTORE
```

```
-----  
BONHOEFFER, DIETRICH  
CROOKES, WILLIAM
```

Questa istruzione cerca una seconda occorrenza della lettera ‘O’ nei nomi degli autori. “>0” è una tecnica logica: si ricordi che le funzioni producono generalmente due tipi diversi di risultati, uno con il quale vengono creati nuovi oggetti e l’altro con il quale vengono fornite informazioni relative a oggetti già esistenti.

Con la funzione **INSTR** vengono fornite informazioni su una stringa, in particolare la posizione del gruppo di caratteri ricercato. In questo caso, viene richiesto di localizzare la seconda ‘O’ nella stringa Autore. Il risultato è un numero maggiore di zero per quei nomi con almeno due ‘O’, e zero per quei nomi con una o meno ‘O’ (se **INSTR** non trova nulla il risultato è zero). Quindi, la semplice verifica della presenza di un risultato maggiore di zero conferma che la ricerca con **INSTR** della seconda ‘O’ ha avuto successo.

La clausola **where** in cui compare la funzione **INSTR** produce lo stesso risultato di questa istruzione:

```
where Autore LIKE '%O%O%'
```

Si ricordi che il segno di percentuale (%) è un carattere jolly, ovvero sta al posto di qualsiasi elemento, quindi la clausola LIKE indica a SQL di cercare due ‘O’ precedute, separate o seguite da qualsiasi carattere. Si tratta di un esempio probabilmente più semplice da comprendere del precedente con INSTR.

Spesso, in Oracle esistono più modi per ottenere il medesimo risultato. Alcune procedure sono più semplici da comprendere, alcune funzionano più velocemente, alcune sono più appropriate in determinate situazioni e alcune sono semplicemente una scelta di stile personale.

## SOUNDEX

Esiste una funzione di stringa che viene utilizzata quasi esclusivamente in una clausola where: SOUNDEX. Con questa insolita funzione vengono trovate parole che sono simili ad altre, indipendentemente dal modo in cui sono scritte. Si tratta di una caratteristica particolarmente utile quando non si è certi della grafia di una parola o di un nome, ma è adatta solo per i termini della lingua inglese.

La sintassi della funzione SOUNDEX è la seguente:

```
SOUNDEX(stringa)
```

Ecco alcuni esempi del suo utilizzo:

```
select Citta, Temperatura, Condizione from CLIMA
where SOUNDEX(Citta) = SOUNDEX('Sidney');
```

CITTA	TEMPERATURA	CONDIZIONE
SYDNEY	-5	NEVE

```
select Citta, Temperature, Condition from CLIMA
where SOUNDEX(Citta) = SOUNDEX('menncestr');
```

CITTA	TEMPERATURA	CONDIZIONE
MANCHESTER	19	SOLE

```
select Autore from RIVISTA;
where SOUNDEX(Autore) = SOUNDEX('Banheffer');
```

AUTORE
BONHOEFFER, DIETRICH

Con la funzione SOUNDEX viene confrontato il suono delle voci inserite nella colonna selezionata con il suono della parola indicata tra apici e viene ricercata una corrispondenza. Si fanno delle ipotesi sulle modalità di pronuncia delle lettere e delle combinazioni di lettere e le due parole confrontate devono iniziare con la stessa lettera. Non sempre viene trovata la parola ricercata, o la parola scritta male, ma talvolta la funzione può risultare d'aiuto.

Non è necessario che una delle due impostazioni di SOUNDEX nella clausola where sia un nome. SOUNDEX può essere utilizzata anche per confrontare due colonne, al fine di trovare quelle simili.

Una finalità utile di questa funzione è quella di ripulire le mailing list. Molti elenchi hanno voci doppie con leggere differenze di grafia o di formato dei nomi dei clienti. Utilizzando la

funzione SOUNDEX per elencare tutti i nomi simili, molti dei duplicati possono essere scoperti ed eliminati.

Si può provare con la tabella INDIRIZZO:

```
select Cognome, Nome, Telefono
  from INDIRIZZO;
```

COGNOME	NOME	TELEFONO
BAILEY	WILLIAM	213-293-0223
ADAMS	JACK	415-453-7530
SEP	FELICIA	214-522-8383
DE MEDICI	LEFTY	312-736-1166
DEMIURGE	FRANK	707-767-8900
CASEY	WILLIS	312-684-1414
ZACK	JACK	415-620-6842
YARROW	MARY	415-787-2178
WERSCHKY	ARNY	415-235-7387
BRANT	GLEN	415-526-7512
EDGAR	THEODORE	415-525-6252
HARDIN	HUGGY	617-566-0125
HILD	PHIL	603-934-2242
LOEBEL	FRANK	202-456-1414
MOORE	MARY	718-857-1638
SZEP	FELICIA	214-522-8383
ZIMMERMAN	FRED	503-234-7491

Per trovare le voci doppie, occorre indurre Oracle a confrontare ogni cognome nella tabella con tutti gli altri presenti nella stessa tabella.

Si crei un alias della tabella INDIRIZZO e si la chiami prima a e poi b. Ora, è come se ci fossero due tabelle, a e b, con la colonna comune Cognome.

Nella clausola where, qualsiasi colonna in cui il cognome in una tabella è identico al cognome nell'altra viene eliminata. Questo serve a impedire che venga trovata la corrispondenza di un cognome con se stesso.

Vengono quindi selezionati i cognomi che appaiono simili:

SOUNDEX(b.Cognome);	COGNOME	NOME	TELEFONO
	SZEP	FELICIA	214-522-8383
	SEP	FELICIA	214-522-8383

Si possono anche impostare ricerche di termini singoli all'interno di una voce di testo. Per esempi di queste e altre ricerche di testo complesse, si rimanda al Capitolo 24.

## Supporto alle lingue nazionali

In Oracle, non è necessario utilizzare caratteri inglesi; i dati possono essere rappresentati in qualsiasi lingua mediante l'implementazione del *National Language Support*. Utilizzando caratteri formati da parti di informazioni più lunghe rispetto ai caratteri usuali, in Oracle è possibile rappresentare stringhe in giapponese o con caratteri analoghi. Si consultino le voci NLSSORT, NLS\_INITCAP, NLS\_LOWER e NLS\_UPPER nel Capitolo 42. Oltre alla funzione SUBSTR,

Oracle supporta anche SUBSTRB (che usa i byte al posto dei caratteri), SUBSTRC (che usa i caratteri Unicode completi), SUBSTR2 (che usa i codepoint UCS2) e infine SUBSTR4 (che invece usa i codepoint UCS4).

## 7.7 Conclusioni

I dati possono essere di vari tipi, in primo luogo DATE, NUMBER e CHARACTER. I dati carattere sono fondamentalmente stringhe di lettere, numeri e altri simboli e vengono spesso definiti “stringhe di caratteri”, o semplicemente “stringhe”. Queste stringhe possono essere modificate o descritte utilizzando funzioni di stringa. In Oracle sono disponibili due tipi di dati carattere: stringhe a lunghezza variabile (tipo di dati VARCHAR2) e stringhe a lunghezza prefissata (tipo di dati CHAR). I valori nelle colonne CHAR vengono riempiti con spazi in modo da occupare la lunghezza completa della colonna, se il testo inserito risulta più corto della lunghezza di colonna definita.

Funzioni come RPAD, LPAD, LTRIM, RTRIM, TRIM, LOWER, UPPER, INITCAP e SUBSTR modificano effettivamente il contenuto di una stringa o di una colonna prima della visualizzazione.

Funzioni come LENGTH, INSTR e SOUNDDEX descrivono le caratteristiche di una stringa, come la lunghezza, la posizione di un certo carattere al suo interno o la somiglianza con altre stringhe.

Tutte queste funzioni possono essere utilizzate da sole o in combinazione per selezionare e presentare le informazioni presenti in un database Oracle. Si tratta di procedimenti chiari, costituiti da fasi logiche semplici che possono essere combinate per ottenere risultati molto sofisticati.



## Capitolo 8

# Giocare con i numeri

- 8.1    **Le tre classi di funzioni numeriche**
- 8.2    **Notazione**
- 8.3    **Funzioni per valori singoli**
- 8.4    **Funzioni di gruppo**
- 8.5    **Funzioni di elenco**
- 8.6    **Ricerca di righe con MAX o MIN**
- 8.7    **Precedenza e parentesi**
- 8.8    **Conclusioni**

**Q**ualsiasi cosa si faccia, in particolar modo in ambito economico, viene misurata, spiegata e spesso guidata dai numeri. Oracle non può rimediare a questa ossessione per i numeri e all'illusione di controllo che spesso l'accompagna, però agevola l'analisi accurata e completa delle informazioni in un database. Una buona analisi dei numeri spesso consente di rivelare tendenze e fatti che inizialmente non risultano evidenti.

## 8.1 Le tre classi di funzioni numeriche

Le funzioni di Oracle si applicano a tre classi di numeri: valori singoli, gruppi di valori ed elenchi di valori. Come per le funzioni di stringa (trattate nel Capitolo 7), con alcune di queste funzioni i valori vengono modificati, mentre con altre vengono semplicemente fornite delle informazioni su di essi. Le classi vengono distinte nel modo seguente.

Un *valore singolo* è un numero, come:

- un numero vero e proprio, per esempio 544.3702;
- una variabile SQLPLUS o PL/SQL;
- un numero tratto da una colonna o da una riga del database.

Con le funzioni di Oracle per i valori singoli questi valori in genere vengono modificati mediante un'operazione di calcolo.

Un *gruppo di valori* corrisponde a tutti i numeri di una colonna tratti da una serie di righe, per esempio il prezzo di chiusura per tutte le righe dei titoli contenuti nella tabella AZIONE, descritta nel Capitolo 1. Le funzioni di Oracle per i gruppi di valori forniscono informazioni sull'intero gruppo, come prezzi medi, e non sui singoli membri del gruppo.

Un *elenco di valori* è una serie di numeri che può comprendere:

- numeri veri e propri, come 1, 7.3, 22, 86;
- variabili SQLPLUS o PL/SQL;
- colonne, come PrezzoApertura, PrezzoChiusura, Offerta, Richiesta.

Le funzioni di Oracle per gli elenchi di valori consentono di operare su un valore tratto dall'elenco.

La Tabella 8.1 mostra queste funzioni divise per classe. Alcune di esse rientrano in più di una classe, mentre altre possono essere classificate sia come funzioni di stringa sia come funzioni numeriche, oppure vengono utilizzate per convertire i dati da una tipologia all'altra. Il Capitolo 10 è dedicato a questo tipo di funzioni.

## 8.2 Notazione

Le funzioni sono riportate con questo tipo di notazione:

`FUNZIONE(valore [,opzione])`

La funzione stessa viene indicata con caratteri maiuscoli, mentre i valori e le opzioni sono in carattere corsivo minuscolo. Tutte le volte che il termine *valore* è indicato in questo modo, può rappresentare un numero vero e proprio, il nome di una colonna numerica in una tabella, il risultato di un calcolo oppure una variabile. Dato che in Oracle non è consentito utilizzare i numeri come nomi di colonna, un numero vero e proprio non dev'essere racchiuso tra apici (che invece segnalano una stringa vera e propria in una funzione di stringa). Anche i nomi di colonna non devono essere racchiusi tra apici.

In tutte le funzioni compare soltanto una coppia di parentesi. Il valore su cui opera questa funzione, così come le informazioni aggiuntive che è possibile passare alla funzione, devono essere racchiusi tra parentesi.

Alcune funzioni hanno delle *opzioni*, o elementi che non sono indispensabili per il loro funzionamento, ma che possono offrire un maggiore controllo se utilizzate. Le opzioni sono

**Tabella 8.1** Funzioni numeriche di Oracle per classe.

---

FUNZIONI A VALORE SINGOLO	
DEFINIZIONE	FUNZIONE
Addizione.	valore1 + valore2
Sottrazione.	valore1 - valore2
Moltiplicazione.	valore1 * valore2
Divisione.	valore1 / valore2
Valore assoluto.	ABS(valore)
Arcocoseno del valore, in radianti.	ACOS(valore)
Arcoseno del valore, in radianti.	ASIN(valore)
Arcotangente del valore, in radianti.	ATAN(valore)
ATAN2 restituisce l'arcotangente di due valori. I valori di input sono infiniti, gli output sono espressi in radianti.	ATAN2(valore1, valore2)
Operazione di calcolo AND sui bit di valore1 e valore2, che devono entrambi risolversi in numeri non negativi. Restituisce un intero.	BITAND(valore1, valore2)

(segue)

**Tabella 8.1** Funzioni numeriche di Oracle per classe. (*continua*)**FUNZIONI A VALORE SINGOLO****FUNZIONE** **DEFINIZIONE**

CEIL(valore)	Il minimo intero maggiore o uguale a valore.
COS(valore)	Coseno di valore.
COSH(valore)	Coseno iperbolico di valore.
EXP(valore)	$e$ elevato all'esponente valore.
FLOOR(valore)	Il massimo intero minore o uguale a valore.
LN(valore)	Logaritmo naturale di valore.
LOG(valore)	Logaritmo in base 10 di valore.
MOD(valore, divisore)	Modulo.
NVL(valore, sostituto)	Sostituto di valore se valore è NULL.
POWER(valore, esponente)	valore elevato a un esponente.
ROUND(valore, precisione)	Arrotondamento di valore a precisione.
SIGN(valore)	1 se valore è positivo, -1 se negativo, 0 se zero.
SIN(valore)	Seno di valore.
SINH(valore)	Seno iperbolico di valore.
SQRT(valore)	Radice quadrata di valore.
TAN(valore)	Tangente di valore.
TANH(valore)	Tangente iperbolica di valore.
TRUNC(valore, precisione)	Valore troncato a precisione.
VSIZE(valore)	Dimensione di memorizzazione di valore in ORACLE.

**FUNZIONI DI GRUPPO****FUNZIONE** **DEFINIZIONE**

AVG(valore)	Media di valore.
CORR(valore1, valore2)	Coefficiente di correlazione di un gruppo di numeri pari.
COUNT(valore)	Conteggio di righe per colonna.
COVAR_POP(valore1, valore2)	Covarianza della popolazione di un gruppo di numeri pari.
COVAR_SAMP(valore1, valore2)	Covarianza del campione di un gruppo di numeri pari.
CUME_DIST(valore)	Distribuzione cumulativa di un valore in un gruppo di valori.

(segue)

**Tabella 8.1** Funzioni numeriche di Oracle per classe. (*continua*)

FUNZIONI DI GRUPPO	
DEFINIZIONE	FUNZIONE
Posto in classifica di una riga in un gruppo ordinato di righe.	DENSE_RANK
Funzione analitica eseguita sulla riga al primo posto in un gruppo.	FIRST( <i>valore</i> )
Distingue i gruppi duplicati risultanti da una specifica GROUP BY.	GROUP_ID( <i>valore</i> )
Utilizzata in combinazione con le funzioni ROLLUP e CUBE per individuare i valori NULL.	GROUPING( <i>espressione</i> )
Restituisce un numero corrispondente al vettore del bit GROUPING associato con una riga.	GROUPING_ID
Funzione analitica eseguita sulla riga all'ultimo posto in un gruppo.	LAST( <i>valore</i> )
Massimo di tutti i <i>valore</i> per gruppi di righe.	MAX( <i>valore</i> )
Minimo di tutti i <i>valore</i> per gruppi di righe.	MIN( <i>valore</i> )
Calcola la classificazione PERCENTILE di un valore in un insieme presupponendo un modello lineare continuo.	PERCENTILE_CONT( <i>valore</i> )
Calcola la classificazione PERCENTILE di un valore in un insieme presupponendo un modello di distribuzione discreto.	PERCENTILE_DISC( <i>valore</i> )
Calcola la classificazione PERCENTILE di un valore in un insieme.	PERCENTILE_RANK( <i>valore</i> )
Calcola la classificazione di un valore in un gruppo di valori.	RANK( <i>valore</i> )
Deviazione standard di tutti i valori per gruppi di righe.	STDDEV( <i>valore</i> )
Deviazione standard della popolazione.	STDDEV_POP( <i>valore</i> )
Deviazione standard del campione.	STDDEV_SAMP( <i>valore</i> )
Somma di tutti i <i>valore</i> per gruppi di righe.	SUM( <i>valore</i> )
Varianza della popolazione.	VAR_POP( <i>valore</i> )
Varianza del campione.	VAR_SAMP( <i>valore</i> )
Varianza di tutti i <i>valore</i> per gruppi di righe.	VARIANCE( <i>valore</i> )
FUNZIONI DI ELENCO	
DEFINIZIONE	FUNZIONE
Restituisce il primo valore non NULL nell'elenco delle espressioni.	COALESCE( <i>valore1, valore2, ...</i> )
Il <i>valore</i> più grande di un elenco.	GREATEST( <i>valore1, valore2, ...</i> )
Il <i>valore</i> più piccolo di un elenco.	LEAST( <i>valore1, valore2, ...</i> )

sempre visualizzate tra parentesi quadre [ ]. Le parti indispensabili di una funzione sono sempre inserite prima delle parti facoltative.

## 8.3 Funzioni per valori singoli

La maggior parte delle funzioni per valori singoli è decisamente semplice. Questo paragrafo fornisce alcuni brevi esempi delle funzioni principali, illustrando sia i risultati delle funzioni, sia la loro corrispondenza con colonne, righe ed elenchi. Dopo gli esempi, viene illustrata la procedura per combinare queste funzioni. La sintassi di tutte queste funzioni è disponibile nel Capitolo 42.

È stata creata una tabella denominata MATEMATICA per mostrare gli effetti di calcolo di molte funzioni matematiche. È composta soltanto da quattro righe e quattro colonne, come mostrato di seguito:

```
select Nome, Sopra, Sotto, Vuoto from MATEMATICA;
```

NOME	SOPRA	SOTTO	VUOTO
NUMERO INT	11	-22	
DECIMALE INF	33.33	-44.44	
DECIMALE MED	55.5	-55.5	
DECIMALE SUP	66.666	-77.777	

Questa tabella è utile perché presenta valori con caratteristiche diverse, che sono indicate dai nomi scelti per le righe. La riga NUMERO INT non contiene decimali. La riga DECIMALE INF contiene decimali minori di 0.5, la riga DECIMALE MED contiene decimali uguali a 0.5 e la riga DECIMALE SUP contiene decimali maggiori di 0.5. Questa gamma è particolarmente importante quando si utilizzano le funzioni ROUND e TRUNC e per comprendere come queste influiscono sul valore di un numero.

A destra della colonna Nome sono presenti altre tre colonne: Sopra, che contiene soltanto i numeri sopra lo zero (numeri positivi); Sotto, che contiene soltanto i numeri sotto lo zero, e Vuoto, che contiene valori NULL.

**NOTA** In Oracle, una colonna numerica può anche non contenere alcun valore al suo interno: quando contiene valori NULL non significa che vi sono valori pari zero, è semplicemente vuota. Questo fatto ha conseguenze importanti nell'esecuzione dei calcoli, come illustrato più avanti.

Non tutte le righe di questa tabella MATEMATICA sono necessarie per dimostrare il funzionamento della maggior parte delle funzioni matematiche; negli esempi a seguire viene utilizzata soprattutto l'ultima riga, DECIMALE SUP. Inoltre, è stato utilizzato il comando column di SQLPLUS per mostrare esplicitamente la precisione dei calcoli, in modo che siano ben visibili i risultati delle funzioni che influiscono sulla precisione di un numero. Per rivedere i comandi SQL e SQLPLUS che hanno prodotto questi risultati, si osservi il file math.sql (disponibile nel CD-ROM allegato al libro). Nel Capitolo 14 vengono trattate le opzioni di formattazione dei numeri.

### Addizione, sottrazione, moltiplicazione e divisione (+, -, \* e /)

La query che segue illustra ciascuna delle quattro funzioni aritmetiche fondamentali utilizzando le colonne Sopra e Sotto:

```
select Nome, Sopra, Sotto, Vuoto,
       Sopra + Sotto  Piu,
       Sopra - Sotto  Meno,
       Sopra * Sotto  Per,
       Sopra / Sotto  Diviso
  from MATEMATICA where Nome = 'DECIMALE SUP';
```

NOME	SOPRA	SOTTO	VUOTO	PIÙ	MENO	PER	DIVISO
DECIMALE SUP	66.666	-77.777		-11.111	144.443	-5185.081482	- .857143

## NULL

Nell'esempio seguente, le stesse quattro operazioni aritmetiche vengono eseguite nuovamente, questa volta utilizzando le colonne Sopra e Vuoto al posto di Sopra e Sotto. Si osservi che qualsiasi operazione aritmetica che coinvolga un valore NULL ha come risultato un valore NULL. Le colonne calcolate (colonne i cui valori sono il risultato di un calcolo) Piu, Meno, Per e Diviso sono tutte vuote.

```
select Nome, Sopra, Sotto, Vuoto,
       Sopra + Vuoto  Piu,
       Sopra - Vuoto  Meno,
       Sopra * Vuoto  Per,
       Sopra / Vuoto  Diviso
  from MATEMATICA where Nome = 'DECIMALE SUP';
```

NOME	SOPRA	SOTTO	VUOTO	PIÙ	MENO	PER	DIVISO
DECIMALE SUP	66.666	-77.777					

Risulta evidente che un valore NULL non può essere utilizzato in un calcolo. NULL non coincide con zero: può essere considerato come un valore sconosciuto. Per esempio, si supponga di avere una tabella con i nomi dei propri amici e la loro età, ma che la colonna Eta per PAT SMITH sia vuota, perché la sua età è sconosciuta. Qual è la differenza tra l'età dell'amica e la propria? Non è ovviamente la propria età meno zero. Un valore noto meno un valore sconosciuto dà come risultato un valore sconosciuto, ovvero NULL. Non è quindi possibile inserire una risposta, perché non esiste. Dato che il calcolo è impossibile, la risposta è NULL.

Questa è anche la ragione per cui non è possibile utilizzare NULL con un segno di uguale in una clausola where (Capitolo 3). Non ha senso dire che, se  $x$  è sconosciuto e  $y$  è sconosciuto,  $x$  e  $y$  sono uguali. Se le età del Sig. Rossi e del Sig. Bianchi sono sconosciute, non significa che i due signori abbiano la stessa età.

Esistono anche casi in cui NULL significa “irrilevante”, come può essere un numero di appartamento in una casa. In alcuni casi, il numero di appartamento potrebbe essere NULL perché è sconosciuto (anche se in realtà esiste), mentre in altri casi risulta NULL semplicemente perché non esiste. I valori NULL verranno esaminati in maniera più dettagliata più avanti in questo capitolo nel paragrafo “NULL nelle funzioni di gruppo”.

## NVL: sostituzione di valori NULL

Nel paragrafo precedente è stato illustrato il concetto generale di NULL, che rappresenta un valore sconosciuto o irrilevante. In casi particolari, tuttavia, anche se un valore risulta sconosciu-

to, è possibile immaginare un'ipotesi ragionevole. Per esempio, nel caso di un corriere, se il 30 per cento degli spedizionieri che ne richiedono i servizi non sono in grado di indicare il peso o il volume delle merci da consegnare, sarebbe assolutamente impossibile per lui valutare il numero degli aerei cargo necessari? Naturalmente no. Sa per esperienza quant'è il peso e il volume medio dei pacchi, quindi considera questi valori ipotetici per i clienti che non hanno fornito le informazioni necessarie. Ecco le informazioni fornite dai clienti:

```
select Cliente, Peso from CONSEGNA;
```

CLIENTE	PESO
JOHNSON TOOL	59
DAGG SOFTWARE	27
TULLY ANDOVER	

Ed ecco come opera la funzione NVL:

```
select Cliente, NVL(Peso,43) from CONSEGNA;
```

CLIENTE	NVL(PESO,43)
JOHNSON TOOL	59
DAGG SOFTWARE	27
TULLY ANDOVER	43

In questo caso, si sa che il peso medio dei pacchi equivale a 43 chili, quindi viene utilizzata la funzione NVL per inserire il valore 43 ogni volta che il pacco di un cliente ha un peso sconosciuto, e quindi dove il valore nella colonna è NULL. Nell'esempio, TULLY ANDOVER non ha fornito il peso del pacco, ma è comunque possibile calcolarlo e ottenere una valutazione attendibile.

La sintassi per la funzione NVL è la seguente:

`NVL(valore, sostituto)`

Se *valore* è un valore NULL, la funzione equivale al valore *sostituto*. Se *valore* non è NULL, la funzione equivale a *valore*. *sostituto* può essere un numero vero e proprio, un'altra colonna o un calcolo. Nel caso del corriere, si potrebbe anche impostare un join di tabella nell'istruzione select in cui il valore di *sostituto* sia tratto da una vista contenente il calcolo effettivo del peso medio di tutti i pacchi di valore noto.

L'utilizzo della funzione NVL non è limitato ai numeri, in quanto si estende a tipi di dati CHAR, VARCHAR2, DATE e altri ancora; tuttavia *valore* e *sostituto* devono appartenere alla stessa tipologia. La funzione risulta veramente utile soltanto nei casi in cui il dato è sconosciuto, non quando è irrilevante.

## ABS: valore assoluto

Il *valore assoluto* è la misura della portata, o della grandezza, di qualcosa. Per esempio, nel mutamento della temperatura o di un indice di borsa, la portata del cambiamento ha valore in sé, indipendentemente dalla direzione (che a sua volta è importante nel suo ambito). Il valore assoluto è sempre un numero positivo.

La sintassi della funzione ABS è la seguente:

$\text{ABS}(\text{valore})$

Si osservino questi esempi:

146  
 $\text{ABS}(-30) = 30$

## CEIL

La funzione CEIL (abbreviazione di “ceiling”, “soffitto”) produce semplicemente l’*intero* (o numero intero) più piccolo che è maggiore o uguale a un valore specificato. Occorre fare molta attenzione all’effetto di questa funzione su numeri negativi.

Nei listati seguenti vengono illustrati la sintassi della funzione CEIL e alcuni esempi:

$\text{CEIL}(\text{valore})$

$\text{CEIL}(2) = 2$   
 $\text{CEIL}(1.3) = 2$   
 $\text{CEIL}(-2) = -2$   
 $\text{CEIL}(-2.3) = -2$

## FLOOR

La funzione FLOOR (lett. “pavimento”) è ovviamente l’opposto della funzione CEIL. Nei listati seguenti vengono illustrati la sintassi della funzione FLOOR e alcuni esempi:

$\text{FLOOR}(\text{valore})$

$\text{FLOOR}(2) = 2$   
 $\text{FLOOR}(1.3) = 1$   
 $\text{FLOOR}(-2) = -2$   
 $\text{FLOOR}(-2.3) = -3$

## MOD

La funzione MOD (“modulo”) è una piccola e antica funzione utilizzata soprattutto nell’elaborazione dei dati per scopi strani, come “controllare le cifre”, operazione che aiuta a garantire la trasmissione accurata di una stringa di numeri. La funzione MOD divide un valore per un divisore e fornisce il resto. Per esempio,  $\text{MOD}(23,6) = 5$  significa dividere 23 per 6. La risposta è 3 con resto di 5, quindi il risultato dell’operazione modulo è 5.

La sintassi della funzione MOD è la seguente:

$\text{MOD}(\text{valore}, \text{divisore})$

Sia *valore* sia *divisore* possono essere un qualunque numero reale. Il valore di MOD è zero se *divisore* è zero o negativo. Si osservino gli esempi seguenti:

$\text{MOD}(100,10) = 0$   
 $\text{MOD}(22,23) = 22$   
 $\text{MOD}(10,3) = 1$   
 $\text{MOD}(-30,23,7) = -2.23$   
 $\text{MOD}(4.1,.3) = .2$

Il secondo esempio mostra il comportamento di MOD qualora il divisore sia più grande del dividendo (il numero che viene diviso). Il risultato è il dividendo stesso. Si osservi anche questo caso importante, in cui *valore* è un numero intero:

`MOD(valore,1) = 0`

Ecco un buon test per verificare se un numero è intero.

## POWER

La funzione POWER offre semplicemente la possibilità di elevare un valore a un dato esponente positivo, come mostrato di seguito:

`POWER(valore,esponente)`

<code>POWER(3,2)</code>	=	9
<code>POWER(3,3)</code>	=	27
<code>POWER(-77.777,2)</code>	=	6049.261729
<code>POWER(3,1.086)</code>	=	3.297264
<code>POWER(64,.5)</code>	=	8

L'esponente può essere qualsiasi numero reale.

## SQRT: radice quadrata

In Oracle, è disponibile una funzione di radice quadrata separata che dà risultati equivalenti a quelli ottenuti con la funzione POWER(*valore*,.5):

`SQRT(valore)`

<code>SQRT(64)</code>	=	8
<code>SQRT(66.666)</code>	=	8.16492
<code>SQRT(4)</code>	=	2

La radice quadrata di un numero negativo è un numero immaginario. In Oracle i numeri immaginari non sono supportati, quindi se si cerca di calcolare la radice quadrata di un numero negativo, viene visualizzato un messaggio di errore.

## EXP, LN e LOG

Le funzioni EXP, LN e LOG vengono utilizzate raramente nei calcoli commerciali, ma sono molto comuni in campo scientifico e tecnico. EXP è  $e$  (2.71828183...) elevato alla potenza specificata; LN indica il logaritmo “naturale”, o in base  $e$ , di un valore. Le prime due funzioni sono una l'inversa dell'altra;  $\text{LN}(\text{EXP}(i)) = \text{valore}$ . Nella funzione LOG sono elaborati una base e un valore positivo.  $\text{LN}(\text{valore})$  equivale a  $\text{LOG}(2.71828183,\text{valore})$ .

`EXP(valore)`

<code>EXP(3)</code>	=	20.085537
<code>EXP(5)</code>	=	148.413159

`LN(valore)`

```
LN(3)          = 1.098612
LN(20.085536) = 3
```

`LOG(valore)`

```
LOG(EXP(1),3) = 1.098612
LOG(10,100)   = 2
```

## ROUND e TRUNC

ROUND e TRUNC sono due funzioni a valore singolo correlate. Con la funzione TRUNC vengono troncate o eliminate le cifre decimali di un numero; con la funzione ROUND un numero viene arrotondato a meno di un dato numero di cifre decimali.

La sintassi delle funzioni ROUND e TRUNC è la seguente:

```
ROUND(valore,precisione)
TRUNC(valore,precisione)
```

Nel listato seguente vengono applicate alcune proprietà degne di nota. Innanzi tutto si osservi questo semplice esempio di selezione dalla tabella MATEMATICA con il comando select. Vengono richieste due cifre decimali (contando dal punto decimale verso destra).

```
select Nome, Sopra, Sotto,
       ROUND(Sopra,2),
       ROUND(Sotto,2),
       TRUNC(Sopra,2),
       TRUNC(Sotto,2)
  from MATEMATICA;
```

NOME	SOPRA	SOTTO	ROUND	ROUND	TRUNC	TRUNC
			(SOPRA,2)	(SOTTO,2)	(SOPRA,2)	(SOTTO,2)
NUMERO INT	11	-22	11	-22	11	-22
LOW DECIMAL	33.33	-44.44	33.33	-44.44	33.33	-44.44
MID DECIMAL	55.5	-55.5	55.5	-55.5	55.5	-55.5
HIGH DECIMAL	66.666	-77.777	66.67	-77.78	66.66	-77.77

Soltanto l'ultima riga viene troncata, perché è l'unica con tre cifre dopo il punto decimale. Sia i numeri positivi sia quelli negativi dell'ultima riga sono stati arrotondati o troncati: 66.666 è stato arrotondato a un numero superiore, 66.67, mentre il valore -77.777 è stato arrotondato a un numero inferiore, -77.78 (ancor più negativo). Quando l'arrotondamento viene impostato a zero cifre, si ottiene questo risultato:

```
select Nome, Sopra, Sotto,
       ROUND(Sopra,0),
       ROUND(Sotto,0),
       TRUNC(Sopra,0),
       TRUNC(Sotto,0)
  from MATEMATICA;
```

NOME	SOPRA	SOTTO	ROUND (SOPRA,0)	ROUND (SOTTO,0)	TRUNC (SOPRA,0)	TRUNC (SOTTO,0)
NUMERO INT	11	-22	11	-22	11	-22
LOW DECIMAL	33.33	-44.44	33	-44	33	-44
MID DECIMAL	55.5	-55.5	56	-56	55	-55
HIGH DECIMAL	66.666	-77.777	67	-78	66	-77

Si osservi che il valore decimale .5 è stato arrotondato per eccesso quando 55.5 è diventato 56. In questo modo, viene applicata la più comune convenzione di arrotondamento americana (altre convenzioni arrotondano per eccesso solo se un numero è maggiore di .5). Si possono paragonare questi risultati con quelli ottenuti con le funzioni CEIL e FLOOR. Le differenze sono significative:

ROUND(55.5) = 56	ROUND(-55.5) = -56
TRUNC(55.5) = 55	TRUNC(-55.5) = -55
CEIL(55.5) = 56	CEIL(-55.5) = -55
FLOOR(55.5) = 55	FLOOR(-55.5) = -56

Infine si osservi che sia ROUND sia TRUNC possono operare con cifre decimali negative, a sinistra del punto decimale:

```
select Nome, Sopra, Sotto,
       ROUND(Sopra,-1),
       ROUND(Sotto,-1),
       TRUNC(Sopra,-1),
       TRUNC(Sotto,-1)
  from MATEMATICA;
```

NOME	SOPRA	SOTTO	ROUND (SOPRA,-1)	ROUND (SOTTO,-1)	TRUNC (SOPRA,-1)	TRUNC (SOTTO,-1)
NUMERO INT	11	-22	10	-20	10	-20
LOW DECIMAL	33.33	-44.44	30	-40	30	-40
MID DECIMAL	55.5	-55.5	60	-60	50	-50
HIGH DECIMAL	66.666	-77.777	70	-80	60	-70

L'arrotondamento di un numero negativo può essere molto utile nella produzione di report di tipo economico, in cui i valori della popolazione o di somme di denaro devono essere arrotondati a milioni, miliardi o migliaia di miliardi.

## SIGN

La funzione SIGN è il rovescio della medaglia della funzione ABS. ABS indica la grandezza di un valore ma non il suo segno, SIGN indica il segno di un valore ma non la sua grandezza.

La sintassi della funzione SIGN è la seguente:

`SIGN(valore)`

Esempi: `SIGN(146) = 1` Confrontare con: `ABS(146) = 146`  
`SIGN(-30) = -1` `ABS(-30) = 30`

Il risultato della funzione SIGN applicata al valore 0 è 0:

SIGN(0)=0

La funzione SIGN viene spesso utilizzata insieme con la funzione DECODE. Questa funzione viene descritta nel Capitolo 17.

## SIN, SINH, COS, COSH, TAN, TANH, ACOS, ATAN, ATAN2 e ASIN

Le funzioni trigonometriche seno, coseno, e tangente sono funzioni scientifiche e tecniche non molto utilizzate in operazioni commerciali. Le funzioni SIN, COS e TAN consentono di ottenere i valori delle funzioni trigonometriche standard per un angolo espresso in radianti (gradi moltiplicati per *pi greco* diviso per 180). Con le funzioni SINH, COSH e TANH vengono fornite le funzioni iperboliche per un dato angolo.

SIN(*valore*)

SIN(30\*3.141593/180) = .5

COSH(*valore*)

COSH(0) = 1

Le funzioni ASIN, ACOS e ATAN restituiscono i valori dell'arcoseno, arcocoseno e arcotangente (espressi in radianti) dei valori dati. La funzione ATAN2 restituisce l'arcotangente di due valori. I valori di input sono infiniti; gli output sono espressi in radianti.

## 8.4 Funzioni di gruppo

Si tratta di funzioni statistiche come SUM, AVG, COUNT e di altre funzioni simili che forniscono informazioni su un gruppo di valori considerati nel complesso: l'età media di tutti gli amici nella tabella citata in precedenza, per esempio, oppure il nome della persona più anziana del gruppo, o di quella più giovane, o il numero dei componenti del gruppo e altro ancora. Anche quando una di queste funzioni fornisce informazioni su una singola riga, come il nome della persona più anziana, si tratta comunque di un'informazione definita dalla relazione della riga con il gruppo.

A partire da Oracle9i, è possibile applicare ai dati una vasta gamma di funzioni statistiche avanzate, comprese quelle per la verifica della regressione e per la campionatura. Nei paragrafi che seguono, verranno descritte le funzioni di gruppo più utilizzate; per le altre (tutte elencate nella Tabella 8.1), si rimanda al Capitolo 42.

## NULL nelle funzioni di gruppo

Nelle funzioni di gruppo, i valori NULL vengono trattati in maniera differente rispetto a quanto accade nelle funzioni a valore singolo. Infatti, nelle funzioni di gruppo i valori NULL vengono ignorati e il risultato viene calcolato senza di essi.

Si prenda come esempio la funzione AVG. Si supponga di avere un elenco di 100 amici con le rispettive età. Se si scegliersero 20 nomi a caso e si calcolasse la media delle loro età, quanto varierebbe il risultato rispetto a quello ottenuto prendendo un diverso elenco casuale di altre 20 persone e calcolandone la media, oppure rispetto alla media di tutti e 100? In effetti, le medie di questi tre gruppi sarebbero molto vicine. Il senso è che la funzione AVG in qualche modo non

viene influenzata dalla mancanza di dati, anche se i dati mancanti rappresentano una percentuale elevata rispetto al numero totale dei record disponibili.

**NOTA** *La funzione AVG non è immune alla mancanza di dati, e possono verificarsi casi in cui il calcolo così effettuato risulta significativamente diverso dal valore corretto (come quando i dati mancanti non sono distribuiti in maniera casuale), tuttavia si tratta di casi meno comuni.*

Si può provare a paragonare la relativa ininfluenza della mancanza di dati per la funzione AVG con quello che accade, per esempio, con la funzione SUM. Quanto è vicina al risultato corretto la somma delle età di 20 amici rispetto alla somma di tutte e 100 le età? Non è affatto vicina. Quindi, avendo una tabella di amici in cui soltanto 20 su 100 hanno fornito la loro età e 80 su 100 non hanno fornito alcun dato (NULL), quale sarebbe la statistica più affidabile sull'intero gruppo e meno influenzabile dall'assenza di dati, l'età media dei 20 nominativi calcolata con la funzione AVG, o la somma delle loro età calcolata con la funzione SUM? Si tratta di una questione completamente diversa rispetto alla possibilità di valutare la somma di tutti e 100 sulla base di 20 (in effetti, è esattamente la media dei 20 calcolata con AVG, moltiplicata per 100). Il punto è che, se non si conosce il numero delle righe con dati NULL, è possibile utilizzare il comando seguente per ottenere un risultato abbastanza attendibile:

```
select AVG(Eta) from LISTA;
```

Non è viceversa possibile ottenere un risultato attendibile da questo listato:

```
select SUM(Eta) from LISTA;
```

La stessa verifica sull'attendibilità dei risultati consente di definire in che modo le altre funzioni di gruppo rispondono alla presenza di dati NULL. Le funzioni STDDEV e VARIANCE consentono di calcolare la tendenza centrale; anch'esse sono relativamente insensibili alla mancanza di dati (questo argomento verrà trattato nel paragrafo "STDDEV e VARIANCE," più avanti nel capitolo.)

Le funzioni MAX e MIN consentono di misurare i valori estremi dei dati disponibili. Le funzioni MAX e MIN possono fluttuare ampiamente, mentre AVG rimane relativamente costante: se si aggiunge una persona di 100 anni a un gruppo di 99 persone di 50 anni, l'età media aumenta soltanto a 50.5, ma l'età massima è raddoppiata. Con l'aggiunta di un neonato, l'età media ritorna a 50 anni, ma l'età minima scende a 0. È evidente che i valori NULL mancanti o sconosciuti possono influire significativamente sulle funzioni MAX, MIN e SUM, quindi occorre prestare particolare attenzione quando si usano queste funzioni, soprattutto se una percentuale significativa di dati è NULL.

È possibile creare funzioni in cui venga considerata anche la distribuzione dei dati e la quantità di valori NULL confrontata con la quantità di valori reali e ottenere ipotesi attendibili con MAX, MIN e SUM? La risposta è affermativa, ma tali funzioni sarebbero proiezioni statistiche, in cui le conclusioni su un particolare insieme di dati devono essere rese esplicite. Non si tratta di un compito adeguato per una funzione di gruppo di carattere generale. Alcuni statistici potrebbero sostenere che queste funzioni dovrebbero avere un risultato NULL se è presente un valore NULL, poiché un risultato di altro tipo potrebbe essere fuorviante. In Oracle viene comunque prodotto un risultato, ma spetta all'utente decidere se esso è attendibile oppure no.

La funzione COUNT rappresenta un caso speciale. Infatti, può essere utilizzata con valori NULL fornendo sempre come risultato un numero, mai un valore NULL. La sintassi e l'utilizzo di questa funzione verranno illustrati tra breve, ma per fare un semplice confronto tra COUNT e le

altre funzioni di gruppo, è sufficiente segnalare che con questa funzione vengono contate tutte le righe non NULL di una colonna, oppure vengono contate tutte le righe. In altre parole, se viene richiesto di contare le età di 100 amici, il risultato visualizzato da COUNT è 20 (poiché soltanto 20 su 100 hanno fornito la loro età). Se invece viene richiesto di contare le righe nella tabella di amici senza specificare una colonna, il risultato visualizzato è 100. Un esempio di queste differenze viene illustrato nel paragrafo “DISTINCT nelle funzioni di gruppo” più avanti in questo capitolo.

## Esempi di funzioni a valore singolo e di gruppo

Né le funzioni di gruppo né le funzioni a valore singolo sono particolarmente difficili da comprendere, tuttavia una panoramica di carattere pratico sulle modalità di funzionamento di ciascuna funzione può essere utile per mettere in evidenza alcune delle opzioni e delle conseguenze del loro utilizzo.

Nella tabella COMFORT degli esempi seguenti sono contenuti i dati fondamentali relativi alla temperatura ordinati per città a mezzogiorno e a mezzanotte, registrati in ognuna delle quattro giornate campione di ciascun anno: gli equinozi (il 21 marzo e il 22 settembre circa) e i solstizi (il 22 giugno e il 22 dicembre circa). Si dovrebbe riuscire a caratterizzare le città sulla base delle temperature riscontrate in queste giornate ogni anno.

Per lo scopo di questi esempi, la tabella ha soltanto otto righe: i dati rilevati nelle quattro giornate del 2001 per le città di San Francisco in California, e Keene nel New Hampshire. Si possono utilizzare le funzioni numeriche di Oracle per analizzare queste città, la loro temperatura media, la variabilità della temperatura e così via, per il 2001. Con i dati relativi a più anni per più città, si potrebbe ottenere un’analisi dei modelli di temperatura e della variabilità per tutto il secolo.

La tabella è di questo tipo:

```
describe COMFORT
```

Nome	Null?	Type
Citta	NOT NULL	VARCHAR2(13)
ESEMCDATA	NOT NULL	DATE
MEZZOGIORNO		NUMBER(3,1)
MEZZANOTTE		NUMBER(3,1)
PRECIPITAZIONE		NUMBER

E contiene i dati di temperatura seguenti:

```
select Citta, DataCampione, Mezzogiorno, Mezzanotte from COMFORT;
```

CITTA	DATA	CAMP	MEZZOGIORNO	MEZZANOTTE
SAN FRANCISCO	21-MAR-01		62.5	42.3
SAN FRANCISCO	22-GIU-01		51.1	71.9
SAN FRANCISCO	23-SET-01			61.5
SAN FRANCISCO	22-DIC-01		52.6	39.8
KEENE	21-MAR-01		39.9	-1.2
KEENE	22-GIU-01		85.1	66.7
KEENE	23-SET-01		99.8	82.6
KEENE	22-DIC-01		-7.2	-1.2

## AVG, COUNT, MAX, MIN e SUM

A causa di un'interruzione nell'alimentazione elettrica, la temperatura di S. Francisco a mezzogiorno del 23 settembre non è stata registrata. Le conseguenze di questo fatto possono essere osservate nella query seguente:

```
select AVG(Mezzogiorno), COUNT(Mezzogiorno), MAX(Mezzogiorno), MIN(Mezzogiorno),
SUM(Mezzogiorno)
from COMFORT
where Citta = 'SAN FRANCISCO';
```

AVG(MEZZO)	COUNT(MEZZO)	MAX(MEZZO)	MIN(MEZZO)	SUM(MEZZO)
55.4	3	62.5	51.1	166.2

AVG(Mezzogiorno) è la media delle tre temperature note. COUNT(Mezzogiorno) è il conteggio delle righe contenute nelle colonne Mezzogiorno che non sono NULL. Il significato di MAX e MIN è evidente. SUM(Mezzogiorno) è la somma di tre sole date a causa del valore NULL per il 23 settembre. Si osservi che

```
SUM(MEZZO)
-----
```

è, non a caso, esattamente tre volte il valore di AVG(Mezzogiorno).

## Combinazione di funzioni a valore singolo e funzioni di gruppo

Si supponga di voler conoscere l'entità del cambiamento della temperatura nel corso di una giornata. Si tratta di una misura di *variabilità*. Il primo tentativo per rispondere al quesito potrebbe essere quello di sottrarre la temperatura a mezzanotte dalla temperatura a mezzogiorno:

```
select Citta, DataCampione, Mezzogiorno-Mezzanotte
from COMFORT
where Citta = 'KEENE';
```

CITTÀ	DATACAMP MEZZOGIORNO-M
KEENE	21.MAR-01 41.1
KEENE	22-JUN-01 18.4
KEENE	23-SEP-01 17.2
KEENE	22-DEC-01 -6

Con sole quattro righe da considerare nella tabella, è possibile convertire velocemente (o ignorare) il fastidioso segno meno (-). La variabilità nella temperatura è in realtà la misura della portata del cambiamento, ovvero l'indicazione di quanto la temperatura è cambiata. Non viene compreso il segno del valore, quindi il valore -6 è sbagliato. Se non viene corretto, e viene considerato in un altro calcolo, come il cambiamento medio in un anno, la risposta sarà del tutto errata, come mostrato di seguito:

```
select AVG(Mezzogiorno-Mezzanotte)
  from COMFORT
 where Citta = 'KEENE';
```

```
AVG(MEZZOGIORNO-MEZZANOTTE)
-----
      17.675
```

La risposta corretta richiede un valore assoluto, come mostrato di seguito.

```
select AVG(ABS(Mezzogiorno-Mezzanotte))
  from COMFORT
 where Citta = 'KEENE';
```

```
AVG(ABS(MEZZOGIORNO-MEZZANOTTE))
-----
      20.675
```

Combinando le funzioni in questo modo si segue la stessa tecnica illustrata nel Capitolo 7, nel paragrafo sulle funzioni di stringa. Una funzione completa come

ABS(Mezzogiorno-Mezzanotte)

viene semplicemente inserita in un'altra funzione come suo valore, come in:

AVG(*valore*)

con questo risultato:

AVG(ABS(Mezzogiorno-Mezzanotte))

In questo esempio, sono illustrate funzioni a valore singolo e di gruppo all'opera. Si noti che è possibile inserire funzioni a valore singolo all'interno di funzioni di gruppo. Con le funzioni a valore singolo viene calcolato un risultato per ciascuna riga, mentre con le funzioni di gruppo il risultato viene considerato come valore effettivo della riga. Le funzioni a valore singolo possono essere combinate (*annidate* una all'interno dell'altra) praticamente senza limite. Le funzioni di gruppo possono contenere le funzioni a valore singolo al posto dei propri valori. In effetti, sono in grado di contenere più funzioni a valore singolo al posto dei loro valori.

Come si procede per combinare funzioni di gruppo? Innanzitutto, non ha alcun senso annidarle in questo modo:

```
select SUM(AVG(Mezzogiorno)) from COMFORT;
```

Viene visualizzato il seguente messaggio di errore:

```
ERROR at line 1:
ORA-00978: nested group function without GROUP BY
```

Inoltre, se anche funzionasse, verrebbe prodotto esattamente lo stesso risultato di quanto segue:

```
AVG(Mezzogiorno)
```

poiché il risultato della funzione AVG(Mezzogiorno) è semplicemente un valore singolo. La somma, con la funzione SUM, di un valore singolo è semplicemente quel valore singolo, pertanto non ha nessun senso annidare le funzioni di gruppo. L'eccezione a questa regola si ha nell'utilizzo di group by all'interno dell'istruzione select, infatti il motivo per cui è stato prodotto il messaggio di errore è proprio l'assenza di questo comando. Questo argomento verrà discusso nel Capitolo 11.

Può avere senso sommare, sottrarre, moltiplicare o dividere i risultati di due o più funzioni di gruppo. Per esempio,

```
select MAX(Mezzogiorno) - MIN(Mezzogiorno)
      from COMFORT
     where Citta = 'SAN FRANCISCO';
```

```
MAX(MEZZOGIORNO)-MIN(MEZZOGIORNO)
```

```
-----  
11.4
```

è la misura del cambiamento di temperatura in un anno. In effetti, con un piccolo sforzo in più si potrebbe effettuare un veloce confronto tra San Francisco e Keene:

```
select Citta, AVG(Mezzogiorno), MAX(Mezzogiorno), MIN(Mezzogiorno),
       MAX(Mezzogiorno) - MIN(Mezzogiorno) Varia
      from COMFORT
     group by Citta;
```

CITTÀ	AVG(MEZZOGIORNO)	MAX(MEZZOGIORNO)	MIN(MEZZOGIORNO)	Varia
KEENE	54.4	99.8	-7.2	107
SAN FRANCISCO	55.4	62.5	51.1	11.4

Questa query offre un buon esempio di come sia possibile ricavare informazioni dai dati disponibili: la temperatura media nelle due città è pressoché identica, ma l'enorme sbalzo di temperatura a Keene, paragonato ai valori di San Francisco, è un dato molto significativo sulla variabilità annuale della temperatura nelle due città e sullo sforzo relativo che occorre per vestirsi adeguatamente (o per riscaldare e raffreddare un'abitazione) in una città piuttosto che nell'altra. La clausola group by verrà spiegata in maniera dettagliata nel Capitolo 11. In breve, in questo esempio aveva il compito di vincolare le funzioni di gruppo a operare non sulla tabella completa, bensì sui sottogruppi delle temperature per città.

## STDDEV e VARIANCE

La deviazione standard e la varianza hanno in comune il significato statistico e utilizzano la stessa sintassi delle altre funzioni di gruppo:

```
select MAX(Mezzogiorno), AVG(Mezzogiorno), MIN(Mezzogiorno), STDDEV(Mezzogiorno),
       VARIANCE(Mezzogiorno)
      from COMFORT
     where Citta = 'KEENE';
```

```
MAX(MEZZO AVG(MEZZO MIN(MEZZO STDDEV(MEZZO VARIANCE(MEZZO
```

```
-----  
99.8   54.4    -7.2    48.3338  2336.15
```

## DISTINCT nelle funzioni di gruppo

In tutte le funzioni di gruppo è possibile impostare l'opzione DISTINCT o l'opzione ALL. La funzione COUNT rappresenta un buon esempio del funzionamento di queste opzioni.

Ecco la sintassi della funzione COUNT (la barra | significa “oppure”):

```
COUNT([DISTINCT | ALL] valore)
```

Ecco un esempio:

```
select COUNT(DISTINCT Citta), COUNT(Citta), COUNT(*)
from COMFORT;
----- ----- -----
COUNT(DISTINCTCITTÀ) COUNT(CITTA) COUNT(*)
----- ----- -----
2           8           8
```

In questa query sono visualizzati alcuni risultati degni di nota. Innanzi tutto, con l'opzione DISTINCT la funzione COUNT conta soltanto i nomi di città unici. Se si richiedesse di contare le temperature di mezzanotte con questa opzione impostata, il risultato sarebbe 7, poiché due delle otto temperature sono uguali. Quando la funzione COUNT viene utilizzata per Citta ma senza il vincolo impostato con DISTINCT, il risultato è 8.

In questo esempio viene dimostrato che la funzione COUNT può essere applicata anche a una colonna di caratteri. Non viene effettuato un calcolo sui valori presenti nella colonna, come nel caso delle funzioni SUM o AVG; viene semplicemente contato il numero delle righe con un valore nella colonna specificata.

La funzione COUNT presenta anche un'altra proprietà singolare: *valore* può essere un asterisco, ovvero con COUNT è possibile ottenere il numero delle righe di una tabella, a prescindere dal fatto che vi siano colonne specifiche con valori NULL. Le righe vengono contate anche se tutti i campi hanno valore NULL.

Nelle altre funzioni di gruppo non è possibile utilizzare l'asterisco come nella funzione COUNT, né è possibile utilizzare una colonna di caratteri come *valore* (cosa possibile con le funzioni MAX e MIN). Viceversa, in tutte le funzioni di gruppo è possibile utilizzare l'opzione DISTINCT, che vincola ad agire soltanto su valori unici. Con una tabella contenente valori come questi:

```
select Nome, Eta from COMPLEANNO;
```

NOME	ETA
GEORGE	42
ROBERT	52
NANCY	42
VICTORIA	42
FRANK	42

si otterebbe questo risultato:

```
select AVG(DISTINCT Eta) Media,
       SUM(DISTINCT Eta) Totale
  from COMPLEANNO;
```

MEDIA	TOTALE
47	94

che, se si desiderasse conoscere l'età media dei propri amici, non sarebbe la risposta corretta. L'uso dell'opzione DISTINCT in funzioni diverse da COUNT è estremamente raro, tranne forse in alcuni calcoli statistici. Per MAX e MIN si ottiene lo stesso risultato con o senza l'opzione DISTINCT.

L'opzione alternativa a DISTINCT è ALL, che rappresenta l'impostazione predefinita. Con l'opzione ALL si indica a SQL di verificare tutte le righe, anche se un valore viene riprodotto in righe diverse. Non è necessario digitare l'opzione ALL; se non si specifica DISTINCT, viene automaticamente utilizzata ALL.

## 8.5 Funzioni di elenco

Diversamente dalle funzioni di gruppo, che operano su un gruppo di righe, le funzioni di elenco operano su un gruppo di colonne, con valori effettivi o calcolati, all'interno di un'unica riga. In altre parole, le funzioni di elenco consentono di confrontare i valori di ciascuna delle diverse colonne e di prendere in considerazione il valore maggiore o minore. Si consideri la tabella COMFORT, visualizzata di seguito:

```
select Citta, DataCampione, Mezzogiorno, Mezzanotte from COMFORT;
```

CITTA	DATA CAMPIONE	MEZZOGIORNO	MEZZANOTTE
SAN FRANCISCO	21-MAR-01	16.9	5.7
SAN FRANCISCO	22-JUN-01	10.6	22.2
SAN FRANCISCO	23-SEP-01		16.4
SAN FRANCISCO	22-DEC-01	11.4	4.3
KEENE	21-MAR-01	4.4	-18
KEENE	22-JUN-01	29.5	19.3
KEENE	23-SEP-01	37.7	28.1
KEENE	22-DEC-01	-22	-18

Ora si confronti il risultato di questa query con il risultato seguente. Si osservino in particolare i valori dei mesi di giugno e settembre per San Francisco, nonché quelli di dicembre per Keene:

```
select Citta, DataCampione, GREATEST(Mezzanotte,Mezzogiorno) AS Alta,
       LEAST(Mezzanotte,Mezzogiorno) AS Bassa
  from COMFORT;
```

CITTA	DATA CAMPIONE	MEZZOGIORNO	MEZZANOTTE
SAN FRANCISCO	21-MAR-01	16.9	5.7
SAN FRANCISCO	22-JUN-01	22.2	10.6
SAN FRANCISCO	23-SEP-01		
SAN FRANCISCO	22-DEC-01	11.4	4.3
KEENE	21-MAR-01	4.4	-18
KEENE	22-JUN-01	29.5	19.3
KEENE	23-SEP-01	37.7	28.1
KEENE	22-DEC-01	-22	-18

Sulla riga di settembre per San Francisco non viene riportato nessun valore, poiché con le funzioni GREATEST e LEAST non è stato possibile confrontare correttamente una temperatura di mezzanotte effettiva con una temperatura di mezzogiorno sconosciuta. Negli altri due casi, la temperatura di mezzanotte era effettivamente maggiore rispetto alla temperatura di mezzogiorno.

Le sintassi per le funzioni GREATEST e LEAST sono le seguenti:

```
GREATEST(valore1, valore2, valore3...)  
LEAST(valore1, valore2, valore3...)
```

Sia GREATEST sia LEAST possono essere utilizzate con molti valori, che possono essere colonne, numeri veri e propri, calcoli o combinazioni di altre colonne. Le funzioni GREATEST e LEAST possono essere utilizzate anche con colonne di caratteri. Per esempio, è possibile utilizzarle per selezionare i nomi che risultano ultimi (GREATEST) o primi (LEAST) in ordine alfabetico:

```
GREATEST('Bob', 'George', 'Andrew', 'Isaiah') = Isaiah  
LEAST('Bob', 'George', 'Andrew', 'Isaiah') = Andrew
```

A partire da Oracle9i, è possibile utilizzare la funzione COALESCE per valutare più valori come valori non NULL. A partire da una stringa di valori, COALESCE restituirà il primo valore non NULL incontrato; se tutti i valori sono NULL, verrà visualizzato un risultato NULL.

Nella tabella COMFORT, per una delle misurazioni di San Francisco c'è un valore NULL nella colonna Mezzogiorno. La query seguente restituisce:

```
select COALESCE(Mezzogiorno, Mezzanotte)  
  from COMFORT  
 where Citta = 'SAN FRANCISCO';  
  
COALESCE(MEZZOGIORNO,MEZZANOTTE)  
-----  
16.9  
16.9  
16.4  
11.4
```

Nei primi due record dell'output, il valore visualizzato è il valore di Mezzogiorno. Nel terzo record, il valore di Mezzogiorno è NULL quindi al suo posto viene restituito quello di Mezzanotte. Le funzioni DECODE e CASE di Oracle offrono una funzionalità simile, come descritto nel Capitolo 17.

## 8.6 Ricerca di righe con MAX o MIN

Quale città ha la più elevata temperatura mai registrata, e in quale data? La risposta è facile, avendo soltanto otto righe da controllare, ma che cosa accadrebbe con i dati di tutte le città del mondo registrati negli ultimi 50 anni? Per il momento, si presuma che la temperatura più elevata dell'anno sia stata registrata più vicino a mezzogiorno che a mezzanotte. La query seguente non funzionerebbe:

```
select Citta, DataCampione, MAX(Mezzogiorno)  
  from COMFORT;
```

Oracle verifica la colonna Citta e invia questo messaggio di errore:

```
select Citta, DataCampione, MAX(Mezzogiorno)
*
ERROR at line 1:
ORA-00937: not a single-group group function
```

Questo messaggio di errore è poco chiaro. Significa che Oracle ha rilevato un errore nella logica della domanda. Se si richiedono le colonne, significa che si desidera che vengano visualizzate le singole righe; l'utilizzo di MAX, una funzione di gruppo, significa che si desidera visualizzare un risultato di gruppo per tutte le righe. Si tratta di due richieste di diverso tipo: con la prima viene richiesto un insieme di righe, con la seconda una riga calcolata, quindi si innesta un conflitto. Ecco come impostare la query:

```
select Citta, DataCampione, Mezzogiorno
  from COMFORT
 where Mezzogiorno = (select MAX(Mezzogiorno) from COMFORT);
```

CITTA	DATACAMP	MEZZOGIORNO
KEENE	23-SEP-01	37.7

Viene prodotta soltanto una riga. Quindi, si potrebbe pensare che la combinazione di una richiesta per le colonne Citta e DataCampione insieme con la funzione MAX per i valori di Mezzogiorno non sia contraddittoria come illustrato poc'anzi. Ma cosa sarebbe successo se fosse stata richiesta la temperatura minima?

```
select Citta, DataCampione, Mezzanotte
  from COMFORT
 where Mezzanotte = (select MIN(Mezzanotte) from COMFORT);
```

CITTA	DATACAMP	MEZZANOTTE
KEENE	21.MAR-01	-18
KEENE	22-DEC-01	-22

Due righe! Più di una riga soddisfa la richiesta impostata con MIN, quindi si innesta un conflitto cercando di combinare una normale richiesta di colonna con una funzione di gruppo.

È anche possibile utilizzare due subquery, ciascuna con una funzione di gruppo (oppure due subquery, una con una funzione di gruppo e l'altra senza). Si supponga di voler conoscere la temperatura massima e minima a mezzogiorno durante l'anno:

```
select Citta, DataCampione, Mezzogiorno
  from COMFORT
 where Mezzogiorno = (select MAX(Mezzogiorno) from COMFORT)
   or Mezzogiorno = (select MIN(Mezzogiorno) from COMFORT);
```

CITTA	DATACAMP	MEZZOGIORNO
KEENE	23-SEP-01	37.7
KEENE	22-DEC-01	-22

## 8.7 Precedenza e parentesi

Quando in un unico calcolo si utilizza più di un operatore aritmetico o logico, quale viene eseguito prima? E l'ordine impostato conta? Si consideri la query seguente della tabella DUAL (una tabella composta da una colonna e da una riga, messa a disposizione da Oracle):

```
select 2/2/4 from DUAL;
```

```
2/2/4
```

```
-----
.25
```

Quando si introducono delle parentesi, anche se i numeri e le operazioni (divisione) non cambiano, la risposta cambia in maniera notevole:

```
select 2/(2/4) from DUAL;
```

```
2/(2/4)
```

```
-----
4
```

La causa di tutto questo si chiama *precedenza*. La precedenza definisce l'ordine con cui vengono eseguiti i calcoli matematici, non solo in Oracle, ma più in generale in matematica. Le regole sono semplici: le operazioni racchiuse tra parentesi hanno la precedenza assoluta, poi vengono considerate le moltiplicazioni e le divisioni, quindi le addizioni e le sottrazioni. Quando si calcola un'equazione, qualsiasi calcolo impostato tra parentesi viene eseguito per primo. Seguono le moltiplicazioni e le divisioni. Infine, vengono effettuate le addizioni e le sottrazioni. Quando devono essere eseguite operazioni con lo stesso grado di precedenza, si segue l'ordine da sinistra a destra. Ecco alcuni esempi:

$$2*4/2*3 = 12 \quad (\text{equivale a } ((2*4)/2)*3)$$

$$(2*4)/(2*3) = 1.333$$

$$4-2*5 = -6 \quad (\text{equivale a } 4 - (2*5))$$

$$(4-2)*5 = 10$$

Anche AND e OR obbediscono alle regole di precedenza; AND ha il grado di precedenza maggiore. Si osservi il risultato di AND e anche l'ordine da sinistra a destra in queste due query:

```
select * from GIORNALE
where Sezione = 'B' AND Pagina = 1 OR Pagina = 2;
```

ARGOMENTO	S	PAGINA
Meteo	C	2
Vita moderna	B	1
Bridge	B	2

3 rows selected.

```
select * from GIORNALE
where Pagina = 1 OR Pagina = 2 AND Sezione = 'B';
```

ARGOMENTO	S	PAGINA
Notizie	A	1
Sport	D	1
Economia	E	1
Vita moderna	B	1
Bridge	B	2

5 rows selected.

Se si desidera davvero visualizzare la pagina 1 o la pagina 2 nella sezione B, è necessario impostare delle parentesi per superare il grado di precedenza di AND. Le parentesi hanno la precedenza su qualsiasi altra operazione.

```
select * from GIORNALE
where Sezione = 'B' AND (Pagina = 1 OR Pagina = 2);
```

ARGOMENTO	S	PAGINA
Vita moderna	B	1
Bridge	B	2

2 rows selected.

La verità è che anche i programmati e i matematici esperti hanno difficoltà a ricordare quali operazioni vengono eseguite per prime quando scrivono una query o un'equazione. Conviene sempre rendere esplicito l'ordine che si desidera venga seguito da Oracle. Occorre utilizzare le parentesi ogni volta in cui potrebbe esservi il più piccolo rischio di confusione.

## 8.8 Conclusioni

Le funzioni per valore singolo operano sui valori riga per riga. Con le funzioni di elenco vengono confrontate le colonne e ne viene selezionata soltanto una, sempre riga per riga. Le funzioni per valore singolo modificano quasi sempre il valore della colonna cui sono applicate. Ciò non significa, naturalmente, che viene modificato il database dal quale viene tratto il valore, ma soltanto che viene effettuato un calcolo con tale valore e il risultato è diverso dal valore originale.

Le funzioni di elenco non modificano i valori in questo modo, piuttosto selezionano (o visualizzano) il valore maggiore o minore, rispettivamente con le funzioni GREATEST e LEAST, di una serie di valori contenuti in una riga. Sia le funzioni per valore singolo sia quelle di elenco non producono un risultato se incontrano un valore NULL.

Entrambe queste funzioni possono essere utilizzate in tutti i casi in cui può essere utilizzata un'espressione, come nelle clausole select e where.

Con le funzioni di gruppo vengono visualizzate delle informazioni su un intero gruppo di numeri, tutte le righe di un insieme. Le funzioni di questo tipo indicano la media di quei valori, il massimo, il loro numero, o la deviazione standard e così via. Nelle funzioni di gruppo i valori NULL vengono ignorati, e questa caratteristica dev'essere tenuta ben presente quando si impongono delle query su gruppi di valori; diversamente, il rischio di risultati errati è elevato.

Con le funzioni di gruppo è anche possibile visualizzare informazioni sui sottogruppi che fanno parte di una tabella, oppure creare una vista di summary delle informazioni da una o più tabelle. Nel Capitolo 11 vengono forniti ulteriori dettagli su queste caratteristiche aggiuntive.

Infine, il grado di precedenza matematica e logica influenza sull'ordine con il quale vengono valutate le query, e ciò può avere un effetto disastroso sui risultati delle query. È necessario prendere l'abitudine di utilizzare le parentesi per rendere esplicito e facile da comprendere l'ordine desiderato.

## Capitolo 9

# Le date

- 9.1    **Aritmetica delle date**
- 9.2    **ROUND e TRUNC nei calcoli delle date**
- 9.3    **Formattazione di TO\_DATE e TO\_CHAR**
- 9.4    **Date nelle clausole where**
- 9.5    **Gestione di più secoli**
- 9.6    **Uso della funzione EXTRACT**
- 9.7    **Uso dei tipi di dati TIMESTAMP**

**U**no dei pregi di Oracle è la capacità di memorizzare e calcolare le date, nonché il numero di secondi, minuti, ore, giorni, mesi e anni che intercorrono tra le date. Oltre alle funzioni di data basilari, Oracle supporta anche una vasta gamma di funzioni per la conversione dei fusi orari. Oracle ha anche la straordinaria capacità di formattare le date praticamente in tutti modi immaginabili, dal semplice “01.05.02” a “1 maggio nell’anno 776 dal regno di Luigi IX”. Probabilmente molte di queste funzioni di formattazione e calcolo delle date sono utilizzate di rado, ma quelle fondamentali si rivelano sicuramente importanti.

**NOTA** *In molti esempi di questo capitolo si utilizzano date espresse nel formato americano, per mostrare l’applicazione dei codici di formato di ORACLE previsti per tale lingua. I risultati effettivi ottenuti nel proprio sistema dipendono dall’impostazione del National Language Support, il supporto per la localizzazione di ORACLE in lingue diverse dall’inglese.*

### 9.1 Aritmetica delle date

DATE è un tipo di dati Oracle, proprio come VARCHAR2 e NUMBER, e ha delle proprietà specifiche. Il tipo di dati DATE è memorizzato in uno speciale formato interno di Oracle che comprende non soltanto il mese, il giorno e l’anno, ma anche l’ora, i minuti e i secondi. Il vantaggio di avere tutti questi dettagli dovrebbe essere evidente. Se, per esempio, si organizza un servizio di assistenza per i clienti, per ogni chiamata registrata Oracle può memorizzare automaticamente la data e l’ora della chiamata in un’unica colonna DATE. È poi possibile formattare la colonna con i dati DATE in un report in modo che mostri soltanto la data, la data e l’ora, la data l’ora e i minuti, oppure la data l’ora i minuti e i secondi. Per memorizzare i secondi frazionari è possibile utilizzare il tipo di dati TIMESTAMP. Per ulteriori dettagli, fare riferimento al paragrafo “Uso dei tipi di dati TIMESTAMP” più avanti in questo capitolo.

SQLPLUS e SQL riconoscono le colonne di tipo DATE, e capiscono che con queste devono applicare le istruzioni dell’*aritmetica delle date*, non della matematica normale. Per esempio, se si aggiunge 1 ad una data, si ottiene un’altra data: il giorno successivo. Sottraendo una data da un’altra si ottiene un numero: il conteggio dei giorni compresi fra le due date.

Tuttavia, poiché le date di Oracle possono includere le ore, i minuti e i secondi, l’aritmetica delle date può diventare complessa, in quanto Oracle potrebbe segnalare che la differenza fra oggi e domani è pari a .516 giorni (questo verrà spiegato in seguito nel capitolo).

## SYSDATE, CURRENT\_DATE e SYSTIMESTAMP

Oracle si collega al sistema operativo del computer per ottenere la data e l'ora correnti. Tramite una funzione speciale, denominata SYSDATE, queste informazioni sono messe a disposizione dell'utente. SysDate può essere considerata come una funzione che dà sempre come risultato la data e l'ora correnti e che può essere utilizzata ovunque come qualsiasi altra funzione Oracle. Si può anche considerarla come una colonna nascosta o una pseudocolonna che si trova in ogni tabella. In questo esempio, SysDate mostra la data odierna:

```
select SysDate from DUAL;
```

```
SYSDATE
-----
15.MAR.02
```

**NOTA** *DUAL è una piccola ma utile tabella Oracle creata per verificare le funzioni o per eseguire rapidi calcoli. Questa tabella verrà descritta più avanti nel capitolo, nel riquadro “La tabella DUAL per brevi test e calcoli”.*

Una seconda funzione, CURRENT\_DATE, segnala la data di sistema con il fuso orario della sessione:

```
select Current_Date from DUAL;
```

```
CURRENT_D
-----
15-MAR-02
```

Un'altra funzione, SYSTIMESTAMP, segnala la data di sistema nel formato del tipo di dati TIMESTAMP:

```
select SysTimeStamp from DUAL;
```

```
SYSTIMESTAMP
-----
15-MAR-02 02.41.31.000000 PM -05:00
```

Per ulteriori dettagli sul tipo di dati TIMESTAMP, le funzioni e i formati utilizzati per il fuso orario, fare riferimento al paragrafo “Uso dei tipi di dati TIMESTAMP” più avanti nel capitolo. I prossimi paragrafi si concentrano sull'utilizzo del tipo di dati DATE, poiché si tratta del tipo di dati che soddisfa la maggior parte dei requisiti di elaborazione.

### La differenza tra due date

FESTA è una tabella che contiene alcune delle feste nazionali negli Stati Uniti per l'anno 2002:

```
select Festa, Dataeffettiva, DataCelebrata from FESTA;
```

FESTA	DATAEFF	DATACELEBRATA
NEW YEAR DAY	01-JAN-02	01-JAN-02
MARTIN LUTHER KING, JR.	15-JAN-02	17-JAN-02
LINCOLNS BIRTHDAY	12-FEB-02	18-FEB-02

## La tabella DUAL per test e calcoli veloci

**D**UAL è una piccolissima tabella fornita da Oracle con una sola riga e una sola colonna:

```
describe DUAL
```

Name	Null?	Type
DUMMY		VARCHAR(1)

Dal momento che molte funzioni Oracle funzionano sia su colonne che su letterali, con DUAL è possibile vederne il funzionamento utilizzando soltanto letterali. In questi esempi, l'istruzione select non si preoccupa di quali colonne sono nella tabella e una singola riga è sufficiente per illustrare un'operazione. Per esempio, si supponga di voler calcolare velocemente POWER(4,3), ovvero 4 al cubo.

```
select POWER(4,3) from DUAL;
```

```
POWER(4,3)
```

```
-----  
64
```

La colonna reale in DUAL è irrilevante. Questo significa che è possibile fare esperimenti con la formattazione e l'aritmetica delle date utilizzando la tabella DUAL e le funzioni di data per capire come operano. Poi, queste funzioni possono essere applicate alle date effettive nelle tabelle reali.

WASHINGTONS BIRTHDAY	22-FEB-02	18-FEB-02
FAST DAY, NEW HAMPSHIRE	22-FEB-02	22-FEB-02
MEMORIAL DAY	30-MAY-02	27-MAY-02
INDEPENDENCE DAY	04-JUL-02	04-JUL-02
LABOR DAY	02-SEP-02	02-SEP-02
COLUMBUS DAY	12-OCT-02	09-OCT-02
THANKSGIVING	28-NOV-02	28-NOV-02

Quali feste non sono celebrate nella data effettiva dell'anniversario nel 2002? Si può rispondere facilmente a questo quesito sottraendo la DataCelebrata dalla Dataeff. Se la risposta non è zero, le due date sono diverse:

```
select Festa, Dataeffettiva, DataCelebrata  
from Festa  
where DataCelebrata - Dataeff != 0;
```

FESTA	DATAEFF	DATACELEBRATA
MARTIN LUTHER KING, JR.	15-JAN-02	17-JAN-02
LINCOLNS BIRTHDAY	12-FEB-02	18-FEB-02
WASHINGTONS BIRTHDAY	22-FEB-02	18-FEB-02
MEMORIAL DAY	30-MAY-02	27-MAY-02
COLUMBUS DAY	12-OCT-02	14-OCT-02

## Aggiunta di mesi

Se il 22 febbraio è il “Fast Day” nel New Hampshire, forse sei mesi dopo si potrebbe celebrare il “Giorno di festa” (traduzione di “Feast Day”). In tal caso, qual è la data? È sufficiente utilizzare la funzione ADD\_MONTHS, per sommare un *conteggio* di sei mesi, come mostrato di seguito:

```
column GiornoFesta heading "Giorno di festa"  
select ADD_MONTHS(DataCelebrata,6) GiornoFesta  
  from FESTA  
 where Festa like 'FAST%';
```

Giorno di festa

-----  
22-AUG-02

## Sottrazione di mesi

Se la prenotazione del posto per il picnic dev’essere effettuata almeno sei mesi prima del Columbus Day, qual è l’ultimo giorno utile per effettuarla? Occorre considerare la DataCelebrata per il Columbus Day e utilizzare ADD\_MONTHS per aggiungere un *numero negativo* di sei mesi (che equivale a sottrarre i mesi). In questo modo, si ottiene la data che precede di sei mesi il Columbus Day. Poi occorre sottrarre un giorno.

```
column UltimoGiorno heading "Ultimo giorno"  
select ADD_MONTHS(DataCelebrata,-6) - 1 UltimoGiorno  
  from FESTA  
 where Festa = 'COLUMBUS DAY';
```

Ultimo giorno

-----  
13-APR-02

## GREATEST e LEAST

Per ciascuna delle feste spostate al lunedì, viene prima la data reale o quella di celebrazione? La funzione LEAST seleziona la data che viene prima in un elenco di date, colonne o letterali; GREATEST seleziona la data più recente. Queste funzioni sono esattamente le stesse utilizzate con i numeri e le stringhe di caratteri:

```
select Festa, LEAST(Dataeffettiva, DataCelebrata) Primo,  
       Dataeffettiva, DataCelebrata  
  from FESTA  
 where Dataeffettiva - DataCelebrata != 0;
```

FESTA	PRIMO	DATAEFF	CELEBRATA
MARTIN LUTHER KING, JR.	15-JAN-02	15-JAN-02	17-JAN-02
LINCOLNS BIRTHDAY	12-FEB-02	12-FEB-02	18-FEB-02
WASHINGTONS BIRTHDAY	18-FEB-02	22-FEB-02	18-FEB-02
MEMORIAL DAY	27-MAY-02	30-MAY-02	27-MAY-02
COLUMBUS DAY	12-OCT-02	12-OCT-02	14-OCT-02

## Funzioni di data

**E**cco un elenco delle principali funzioni applicabili al tipo di dati DATE.

- **ADD\_MONTHS**(conta,data) Somma conta mesi a data.
- **CURRENT\_DATE** Restituisce la data corrente nel fuso orario della sessione.
- **CURRENT\_TIMESTAMP** Restituisce l'indicatore orario nel fuso orario della sessione.
- **DBTIMEZONE** Restituisce il fuso orario corrente del database, in formato UTC.
- **EXTRACT**(unitàtempo FROM dataora) Estrae una porzione di una data da un valore data, come l'estrazione del valore del mese dai valori data di una colonna.
- **FROM\_TZ(indicatore\_orario)** Converte un valore di indicatore orario in un indicatore orario con il valore di fuso orario.
- **GREATEST**(data1,data2,data3,...) Prende la più recente fra le date elencate.
- **LEAST**(data1,data2,data3,...) Prende la meno recente fra le date elencate.
- **LAST\_DAY**(data) Fornisce la data dell'ultimo giorno del mese a cui appartiene data.
- **LOCALTIMESTAMP** Restituisce l'indicatore orario nel fuso orario attivo, senza mostrare informazioni su quest'ultimo.
- **MONTHS\_BETWEEN**(data2,data1) Fornisce data2–data1 in mesi (può essere un numero frazionario).
- **NEW\_TIME**(data,'questo', 'altro') Fornisce la data (e l'ora) nel fuso orario questo, che dev'essere sostituito da un'abbreviazione di tre lettere per indicare il fuso orario corrente. altro dev'essere sostituito da un'abbreviazione di tre lettere per il fuso orario di cui si desidera conoscere la data e l'ora.

I fusi orari sono i seguenti:

AST/ADT	Standard Atlantico/ora solare.
BST/BDT	Standard di Bering/ora solare.
CST/CDT	Standard Centrale/ ora solare.
EST/EDT	Standard Orientale/ora solare.
GMT	Ora di Greenwich.
HST/HDT	Standard Alaska-Hawaii/ora solare.
MST/MDT	Standard Mountain/ora solare.
NST	Tempo standard di Newfoundland.
PST/PDT	Standard Pacifico/ora solare.
YST/YDT	Standard dello Yukon/ora solare.

- **NEXT\_DAY**(data,'giorno') Fornisce la data del giorno successivo a date, dove 'giorno' è 'Monday', 'Tuesday' e così via.
- **NUMTODSINTERVAL**('valore','unitàdata') Converte valore in un letterale INTERVAL DAY TO SECOND, dove 'unitàdata' è 'DAY', 'HOUR', 'MINUTE' o 'SECOND'.
- **NUMTOYINTERVAL**('valore','unitàdata') Converte valore in un letterale INTERVAL YEAR TO MONTH, dove 'unitàdata' è 'YEAR' o 'MONTH'.
- **ROUND**(data,'formato') Senza un formato specificato arrotonda una data a 12 A.M. (mezzanotte, l'inizio del giorno) se l'ora della data è prima di mezzogiorno; altrimenti, l'arrotonda al giorno successivo. Per l'utilizzo di un formato per l'arrotondamento, consultare la voce "ROUND" nel Capitolo 42.
- **SESSIONTIMEZONE** Restituisce il valore del fuso orario della sessione corrente.
- **SYS\_EXTRACT\_UTC** Estrae il valore Coordinated Universal Time (UTC) dalla data corrente.

- **SYSTIMESTAMP** Restituisce la data di sistema, compresi i secondi frazionari e il fuso orario del database.
- **SYSDATE** Restituisce la data e ora correnti.
- **TO\_CHAR(data,'formato')** Riformatta la data secondo il formato specificato.\*
- **TO\_DATE(stringa,'formato')** Converte la data da un particolare formato in una data Oracle. Accetta anche un numero invece di una stringa, con determinati limiti; il formato è limitato.
- **TO\_DSINTERVAL('valore')** Converte il valore di un tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un tipo INTERVAL DAY TO SECOND.
- **TO\_TIMESTAMP('valore')** Converte il valore di un tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un valore di tipo TIMESTAMP.
- **TO\_TIMESTAMP\_TZ('valore')** Converte il valore di un tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un valore di tipo TIMESTAMP WITH TIMEZONE.
- **TO\_YMINTERVAL('valore')** Converte il valore di un tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un valore di tipo INTERVAL YEAR TO MONTH.
- **TRUNC(data,'formato')** Senza un formato imposta una data a 12 A.M. (mezzanotte, l'inizio del giorno). Per l'uso di un formato per il troncamento, consultare la voce "TRUNC" nel Capitolo 42.
- **TZ\_OFFSET('valore')** Restituisce la differenza di fuso orario corrispondente al valore inserito in base alla data di esecuzione dell'istruzione.

\*Consultare il riquadro "Formati delle date" più avanti in questo capitolo.

In questo caso, LEAST funziona abbastanza bene, perché opera sulle colonne DATE di una tabella. E i letterali?

```
select LEAST('20-JAN-02','20-DEC-02') from DUAL;
```

```
LEAST('20
-----
20-DEC-02
```

Questo risultato è errato, un po' come se al posto di LEAST si fosse detto GREATEST. Il 20 dicembre 2002 non cade prima del 20 gennaio 2002. Perché si è arrivati a questo risultato? Perché la funzione LEAST ha considerato questi letterali come *stringhe*.

Non sapeva di doverli considerare come date. Occorre quindi utilizzare la funzione TO\_DATE per convertire questi letterali in un formato interno DATE che Oracle possa utilizzare per le funzioni orientate alle date:

```
select LEAST( TO_DATE('20-JAN-02'),TO_DATE('20-DEC-02') ) from DUAL;
```

```
LEAST(TO_
-----
20-JAN-02
```

## NEXT\_DAY

NEXT\_DAY calcola la data del prossimo giorno della settimana (ossia, domenica, lunedì, martedì, mercoledì, giovedì, venerdì o sabato) successivo alla data specificata. Per esempio, si sup-

---

## Un avvertimento su GREATEST e LEAST

---

**A**differenza della maggior parte delle funzioni e degli operatori logici di Oracle, GREATEST e LEAST non considerano come date le stringhe di caratteri eventualmente corrispondenti a tale formato. Le date sono trattate come stringhe:

```
select Festa, DataCelebrata
  from FESTA
 where DataCelebrata = LEAST('17-JAN-02', '02-SEP-02');
```

FESTA	DATACELEBRATA
-----	-----
LABOR DAY	02-SEP-02

Per fare in modo che LEAST e GREATEST funzionino correttamente, alle stringhe letterali deve essere applicata la funzione TO\_DATE:

```
select Festa, DataCelebrata
  from FESTA
 where DataCelebrata = LEAST( TO_DATE('17-JAN-02'),
                               TO_DATE('02-SEP-02') );
FESTA           DATACELEBRATA
-----          -----
MARTIN LUTHER KING, JR.    17-JAN-02
```

ponga che il giorno di paga sia sempre il primo venerdì dopo il 15 del mese. La tabella GIORNOPAGA contiene solo le date del periodo di paga, ovvero il 15 del mese, con una riga per ogni mese dell'anno:

```
select DataCiclo from GIORNOPAGA;
```

DATA CICLO
-----
15-JAN-02
15-FEB-02
15-MAR-02
15-APR-02
15-MAY-02
15-JUN-02
15-JUL-02
15-AUG-02
15-SEP-02
15-OCT-02
15-NOV-02
15-DEC-02

Quali sono le date di paga reali?

```
column GiornoPaga heading "Giorno Paga"
```

```
select NEXT_DAY(DataCiclo,'VENERDI') GiornoPaga
  from GIORNOPAGA;
```

Giorno Paga

-----  
18-JAN-02  
22-FEB-02  
22-MAR-02  
19-APR-02  
17-MAY-02  
21-JUN-02  
19-JUL-02  
16-AUG-02  
20-SEP-02  
18-OCT-02  
22-NOV-02  
20-DEC-02

Questo risultato è quasi corretto, tranne per i mesi di febbraio, marzo e novembre, perché `NEXT_DAY` è la data del venerdì *successivo* alla data del periodo di paga. Dal momento che il 15 febbraio, il 15 marzo e il 15 novembre cadono di venerdì, si ottiene (erroneamente) il venerdì seguente. La versione corretta è questa:

```
column GiornoPaga heading "Giorno paga"  
  
select NEXT_DAY(DataCiclo-1,'VENERDI') GiornoPaga  
  from GIORNOPAGA;
```

`NEXT_DAY` è veramente una funzione del tipo “maggiori di”. Richiede la data *maggiori* di quella indicata che cade in un giorno specifico della settimana. Per tenere conto anche delle date che cadono già di venerdì, occorre sottrarre 1 dalla data del periodo di paga. In questo modo, ogni data del periodo di paga compare un giorno prima di `NEXT_DAY`. Quindi, i giorni di paga sono sempre i venerdì giusti.

## LAST\_DAY

Questa funzione restituisce la data dell’ultimo giorno del mese. Si supponga che le commissioni e i premi vengano pagati sempre l’ultimo giorno del mese. Quali sono queste date nel 2002?

```
column FineMese heading "Fine mese"  
  
select LAST_DAY(DataCiclo) FineMese  
  from GIORNOPAGA;  
  
Fine mese  
-----  
31-JAN-02  
28-FEB-02  
31-MAR-02  
30-APR-02  
31-MAY-02  
30-JUN-02  
31-JUL-02  
31-AUG-02  
30-SEP-02  
31-OCT-02  
30-NOV-02  
31-DEC-02
```

## **MONTHS\_BETWEEN due date**

Recentemente è stato trovato un file contenente le date dei compleanni di un gruppo di amici. Si possono caricare le informazioni in una tabella denominata COMPLEANNO e visualizzarle:

```
select Nome, Cognome, DataNascita from COMPLEANNO;
```

NOME	COGNOME	DATANASCITA
GEORGE	SAND	12-MAY-46
ROBERT	JAMES	23-AUG-37
NANCY	LEE	02-FEB-47
VICTORIA	LYNN	20-MAY-49
FRANK	PILOT	11-NOV-42

Per calcolare l'età di ogni persona, occorre calcolare i mesi trascorsi tra la data odierna e le date di nascita, quindi dividere per 12, per ottenere gli anni:

```
select Nome, Cognome, DataNascita,
       MONTHS_BETWEEN(SysDate,DataNascita)/12 Eta
  from COMPLEANNO;
```

NOME	COGNOME	DATANASCITA	ETA
GEORGE	SAND	12-MAY-46	55.8728
ROBERT	JAMES	23-AUG-37	64.5933
NANCY	LEE	02-FEB-47	55.1497
VICTORIA	LYNN	20-MAY-49	52.8509
FRANK	PILOT	11-NOV-42	59.3755

La divisione visualizzerà l'età con un componente decimale. Dato che la maggior parte delle persone sopra i sette anni non indica la propria età utilizzando porzioni di anni, si potrebbe applicare la funzione FLOOR al calcolo.

## **Combinazione di funzioni di data**

Si supponga di essere stati assunti il 15.03.02 in un nuovo posto di lavoro, con uno stipendio iniziale inferiore alle aspettative, ma con la promessa di una revisione dopo sei mesi, il primo giorno del mese successivo. Se la data corrente è il 15.03.02, quando cadrà la data di revisione dello stipendio?

```
select SysDate Oggi,
       LAST_DAY(ADD_MONTHS(SysDate,6)) + 1 Revisione
  from DUAL;
OGGI      REVISIONE
-----
15-MAR-02 01-OCT-02
```

ADD\_MONTHS considera SysDate e vi somma sei mesi. LAST\_DAY considera questo risultato e calcola l'ultimo giorno di quel mese. Poi occorre sommare 1 alla data per ottenere il primo giorno del mese successivo. Quanti giorni mancano a quella revisione? È sufficiente sottrarre da questa data (revisione) la data odierna. Si osservi l'uso delle parentesi per assicurare l'ordine corretto di calcolo:

```
select (LAST_DAY(ADD_MONTHS(SysDate,6))+ 1)-SysDate Attesa
  from DUAL;
```

```
ATTESA
```

```
-----
```

```
200
```

## 9.2 ROUND e TRUNC nei calcoli delle date

Si supponga che questa sia la SysDate odierna:

```
SYSDATE
```

```
-----
```

```
15-MAR-02
```

All'inizio del capitolo si è osservato che Oracle può sottrarre una data dall'altra, per esempio domani meno oggi, e produrre una risposta che non è un numero intero. È opportuno esaminare questa situazione:

```
select TO_DATE('16-MAR-02')- SysDate from DUAL;
```

```
TO_DATE('16-MAR-02')-SYSDATE
```

```
-----
```

```
.516
```

La ragione del numero frazionario di giorni tra oggi e domani è che nelle informazioni di data Oracle mantiene le ore, i minuti e i secondi e SysDate è sempre attuale, aggiornata al secondo. Ovviamente, a domani manca meno di un giorno intero.

Per semplificare alcune delle difficoltà che si possono incontrare nell'utilizzo di frazioni di giorni, Oracle fa alcune ipotesi riguardo le date.

- A una data inserita come letterale, per esempio '16.03.02', è assegnata come ora predefinita 12 A.M. (mezzanotte), l'inizio della giornata.
- Una data inserita attraverso SQLPLUS, a meno che l'ora non sia assegnata specificamente, è impostata a 12 A.M. (mezzanotte) l'inizio della giornata.
- SysDate include sempre sia la data che l'ora, a meno che non venga esplicitamente arrotondata. La funzione ROUND imposta qualsiasi data alle 12 A.M. di quel giorno, se l'ora è prima di mezzogiorno, e a 12 A.M. del giorno successivo se è dopo mezzogiorno. La funzione TRUNC opera in modo simile, solo che imposta l'ora a 12 A.M. fino a un secondo prima della mezzanotte.

Per ottenere il numero arrotondato di giorni tra oggi e domani, si può utilizzare quanto segue:

```
select TO_DATE('16-MAR-02')-ROUND(SysDate) from DUAL;
```

```
TO_DATE('16-MAR-02')-ROUND(SYSDATE)
```

```
-----
```

```
1
```

Se l'ora corrente è dopo mezzogiorno, la differenza arrotondata è di 0 giorni.

ROUND, senza un *formato* specificato (si consulti il paragrafo precedente “Funzioni di data”), arrotonda sempre una data alle 12 A.M. del giorno più vicino. Se le date utilizzate contengono orari diversi dal mezzogiorno, occorre impiegare ROUND o accettare possibili risultati frazionari nei calcoli. TRUNC funziona in modo simile, ma imposta l’ora alle 12 A.M. del giorno corrente.

### 9.3 Formattazione di TO\_DATE e TO\_CHAR

TO\_DATE e TO\_CHAR sono simili in quanto entrambe hanno potenti capacità di formattazione. Ma sono anche opposte, poiché TO\_DATE converte una stringa di caratteri o un numero in una data Oracle, mentre TO\_CHAR converte una data Oracle in una stringa di caratteri. La sintassi di queste due funzioni è la seguente:

```
TO_CHAR(data[, 'formato'[, 'NLSparametri']]])  
TO_DATE(stringa[, 'formato'[, 'NLSparametri']]])
```

*data* dev’essere una colonna definita in Oracle come tipo DATE: non può essere una stringa, neppure se è nel formato data predefinito (GG-MMM-AA). L’unico modo di utilizzare una stringa al posto di *data* nella funzione TO\_CHAR è quello di includerla in una funzione TO\_DATE.

*stringa* è una stringa letterale, un numero vero e proprio o una colonna di un database che contiene una stringa o un numero. In tutti i casi, tranne uno, il formato di *stringa* deve corrispondere a quello descritto da *formato*. Solo se una *stringa* è nel formato predefinito, *formato* può essere tralasciato. Il valore predefinito è DD-MON-YY, ma può essere cambiato con:

```
alter session set NLS_DATE_FORMAT = "DD/MON/YYYY";
```

per una data sessione SQL o con il parametro NLS\_DATE\_FORMAT di init.ora.

*formato* è una collezione di diverse opzioni, che possono essere combinate in un numero praticamente infinito di modi. Queste opzioni con le rispettive spiegazioni sono elencate nel riquadro “Formati delle date”. Una volta compreso il metodo base di utilizzo delle opzioni, la loro applicazione è semplice.

NLSparametri è una stringa che imposta l’opzione NLS\_DATE\_LANGUAGE a una particolare lingua, invece di utilizzare la lingua della sessione SQL corrente. Solitamente, questa opzione non viene utilizzata spesso. Oracle restituisce i nomi del giorno e del mese nella lingua impostata per la sessione con alter session.

**NOTA** In molti casi è possibile usare la funzione EXTRACT al posto di TO\_CHAR. Per vedere degli esempi, fare riferimento al paragrafo “Uso della funzione EXTRACT” più avanti nel capitolo.

Per illustrare un esempio del funzionamento delle opzioni viene utilizzata TO\_CHAR. La prima cosa da fare è definire un formato di colonna per i risultati della funzione TO\_CHAR; infatti, senza di esso, TO\_CHAR produce una colonna in SQLPLUS della grandezza di circa 100 caratteri. Rinominando la colonna (per rendere più leggibile la sua intestazione) e impostando il formato a 30 caratteri si ottiene una visualizzazione pratica:

```
column Formattato format a30 word_wrapped  
select DataNascita, TO_CHAR(DataNascita,'MM/DD/YY') Formattato  
from COMPLEANNO  
where Nome = 'VICTORIA';
```

DATANASCITA FORMATTATO

-----  
20-MAY-49 05/20/49

DataNascita mostra il formato di data predefinito in Oracle: DD-MON-YY o giorno del mese, trattino, tre lettere di abbreviazione per il mese, trattino, ultime due cifre dell'anno. La funzione TO\_CHAR nell'istruzione select è abbastanza chiara. MM, DD e YY nell'istruzione TO\_CHAR sono simboli chiave per Oracle da seguire nella formattazione della data. Le barre (/) sono soltanto segni di interpunkzione, e Oracle ne accetta molti altri. Questi segni non devono avere per forza un senso particolare. Per esempio, di seguito viene utilizzato ‘>’ come segno di interpunkzione:

```
select DataNascita, TO_CHAR(DataNascita,'YYMM>DD') Formattato
  from COMPLEANNO
 where Nome = 'VICTORIA';
```

DATANASCITA FORMATTATO

-----  
20-MAY-49 4905>20

Oltre alla punteggiatura standard, Oracle consente anche di inserire del testo nel *formato*. Si può ottenere questo risultato racchiudendo il testo desiderato tra *virgolette*:

```
select DataNascita, TO_CHAR(DataNascita,'Month, DDth "in, um,"YyyY') Formattato
  from COMPLEANNO ;
```

DATANASCITA FORMATTATO

-----  
12-MAY-46 May . 12TH in, um, 1946  
23-AUG-37 August . 23RD in, um, 1937  
02-FEB-47 February . 02ND in, um, 1947  
20-MAY-49 May . 20TH in, um, 1949  
11-NOV-42 November . 11TH in, um, 1942

È interessante osservare alcune conseguenze del *formato*. La parola intera Month comunica a Oracle di utilizzare nella visualizzazione il nome intero del mese. Dato che è stata specificata con la prima lettera maiuscola e le altre minuscole, ogni mese nel risultato viene formattato allo stesso modo. Le opzioni per il mese sono le seguenti:

<b>Formato</b>	<b>Risultato</b>
Month	August
month	august
Mon	Aug
mon	aug

Il giorno del mese viene ottenuto con DD in *formato*. Un suffisso th aggiunto a DD comunica a Oracle di utilizzare i suffissi *ordinali* inglesi, come “TH”, “RD” e “ND” insieme al numero. In questo caso, anche i suffissi possono essere in maiuscolo o minuscolo, ma questo dipende da DD e non da th:

<b>Formato</b>	<b>Risultato</b>
DDth o DDTH	11TH
Ddth o DdTH	11Th
ddth o ddTH	11th

## Formati delle date

I seguenti formati di date vengono utilizzati sia con TO\_CHAR che con TO\_DATE.

MM	Numero di mesi: 12.
RM	Numerazione romana dei mesi: XII.
MON	Abbreviazione di tre lettere del mese: AUG.
MONTH	nome completo del mese : AUGUST, con riempimento fino a 9 caratteri.
DDD	Numero del giorno nell'anno, da Jan 1: 354.
DD	Numero del giorno del mese: 23.
D	Numero del giorno nella settimana: 6.
DY	Abbreviazione del giorno a tre lettere: FRI.
DAY	Giorno scritto per esteso, con riempimento fino a 9 caratteri.
YYYY	Anno scritto per esteso a quattro cifre: 1946.
Y,YYY	Anno, con la virgola.
SYYYY	Anno con segno, se B.C. 01000 B.C. = -1000.
YYY	Ultime tre cifre dell'anno: 946.
YY	Ultime due cifre dell'anno: 46.
Y	Ultima cifra dell'anno: 6.
IYYY	Standard ISO per l'anno a quattro cifre.*
IYY	Standard ISO per l'anno a tre cifre.
IY	Standard ISO per l'anno a due cifre.
I	Standard ISO per l'anno a una cifra.
RR	Ultime due cifre dell'anno relative alla data corrente.
RRRR	Anno arrotondato, accetta l'input a due o a quattro cifre.
CC	Secolo (20 per il 1999).
SCC	Secolo, con le date BC precedute dal segno -.
YEAR	Anno scritto per esteso: NINETEEN-FORTY-SIX.
SYEAR	Anno, con segno – prima delle date BC.
Q	Numero del trimestre: 3.
WW	Numero della settimana nell'anno, dove la settimana 1 inizia il primo giorno dell'anno.
IW	Numero della settimana dell'anno in base allo standard ISO.
W	Numero della settimana nel mese, dove la settimana 1 inizia il primo giorno del mese.
J	"Giuliano"— giorni dal 31 Dicembre, 4712 B.C.: 2422220.
HH	Ore del giorno, sempre 1-12: 11.
HH12	Identico a HH.
HH24	Ore del giorno, orologio di 24 ore: 17.
MI	Minuto dell'ora: 58.
SS	Secondo del minuto: 43.
SSSSS	Secondi da mezzanotte, sempre 0-86399: 43000.
FF	Secondi frazionari come in HH.MI.SS.FF.

X	Carattere radice locale.
/:-:;	Punteggiatura da incorporare nella visualizzazione per TO_CHAR o ignorata nel formato per TO_DATE.
A.M.	Visualizza A.M. o P.M., a seconda dell'ora del giorno.
P.M.	Stesso effetto di A.M.
AM o PM	Stesso effetto di A.M. ma senza punti.
B.C.	Visualizza B.C. o A.D. a seconda della data.
A.D.	Identico a B.C.
BC o AD	Identico a B.C. ma senza punti.
E	Nome dell'era abbreviato, per i calendari asiatici.
EE	Nome dell'era completo, per i calendari asiatici.
TZD	Informazioni sull'ora legale.
TZH	Ora del fuso orario.
TZM	Minuto del fuso orario.
TZR	Regione del fuso orario.

I formati di data seguenti funzionano solo con TO\_CHAR. Non funzionano con TO\_DATE.

"string"	stringa è incorporata nella visualizzazione per TO_CHAR.
fm	Prefisso per il mese e il giorno: fmMONTH o fmday. Elimina il riempimento per il mese o il giorno (definiti in precedenza) nel formato. Senza fm, tutti i mesi vengono visualizzati con la stessa ampiezza. Stessa cosa per i giorni. Con fm, il riempimento viene eliminato. I mesi e i giorni sono di lunghezza semplicemente pari al conteggio dei caratteri che li compongono.
Fx	Formato esatto: specifica la corrispondenza esatta di formato per il modello dell'argomento carattere e per il formato della data.
TH	Suffisso per un numero: ddTH o DDTH produce 24th o 24TH. Le lettere minuscole o maiuscole dipendono da come viene scritto il numero, DD, non da come viene scritto TH. Funziona con qualsiasi numero di una data: YYYY, DD, MM, HH, MI, SS e così via.
SP	Suffisso per un numero che lo forza a essere scritto per esteso: DDSP, DdSP o ddSP produce THREE, Three o three. La scrittura maiuscola o minuscola dipende da come è scritto il numero, DD, non da come sono scritte le lettere SP. Funziona con qualsiasi numero di una data: YYYY, DD, MM, HH, MI, SS e così via.
SPTH	Suffisso combinazione di TH e SP che forza un numero a essere sia specificato per esteso che ad avere un suffisso ordinale. Mspth produce Third. La scrittura maiuscola o minuscola dipende da come è scritto il numero, DD, non da come sono scritte le lettere SP. Funziona con qualsiasi numero di una data: YYYY, DD, MM, HH, MI, SS e così via.
THSP	Identico a SPTH.

\*ISO è l'International Standards Organization, che ha un set di standard per le date diverso dai formati degli Stati Uniti.

Questo stesso approccio vale per tutti i numeri in *formato*, compreso il secolo, l'anno, il trimestre, il mese, la settimana, il giorno del mese (DD), il giorno juliano, le ore, i minuti e i secondi.

Le parole tra virgolette vengono semplicemente inserite dove si trovano. Gli spazi tra qualsiasi di queste richieste di formato sono riprodotti anche nel risultato (si osservino i tre spazi nell'esempio precedente prima della parola "in"). YyyY è stato inserito semplicemente per dimostrare che non c'è distinzione tra maiuscole e minuscole, a meno che non venga utilizzato un suffisso come Th. Per semplicità, si consideri questa richiesta di formattazione:

```
select DataNascita, TO_CHAR(DataNascita,'Month, ddth, YyyY')
      Formattato
     from COMPLEANNO;
DATANASCITA FORMATTATO
-----
12-MAY-46  May      , 12th, 1946
23-AUG-37  August   , 23rd, 1937
02-FEB-47  February , 02nd, 1947
20-MAY-49  May      , 20th, 1949
11-NOV-42  November , 11th, 1942
```

Si tratta di un formato abbastanza normale. I giorni sono tutti allineati, per cui è facile confrontare le righe. Questo è l'allineamento predefinito, e Oracle lo realizza aggiungendo alla destra dei nomi dei mesi degli spazi fino a raggiungere la larghezza di nove. Ci sono casi dove è più importante che una data sia formattata normalmente, per esempio all'inizio di una lettera. Nell'esempio, gli spazi tra mese e virgola sembrerebbero strani. Per eliminarli, si utilizza fm come prefisso per le parole month o day:

<b>Formato</b>	<b>Risultato</b>
Month, ddth	August, 20th
fmMonth, ddth	August, 20th
Day, ddth	Monday, 20th
fmDay, ddth	Monday, 20th

Si consideri l' esempio seguente:

```
select DataNascita, TO_CHAR(DataNascita,'fmMonth, ddth, YyyY')
      Formattato
     from COMPLEANNO;
DATANASCITA FORMATTATO
-----
12-MAY-46  May, 12th, 1946
23-AUG-37  August, 23rd, 1937
02-FEB-47  February, 2nd, 1947
20-MAY-49  May, 20th, 1949
11-NOV-42  November, 11th, 1942
```

Combinando tutti questi controlli di formato e aggiungendo le ore e i minuti, si può produrre un certificato di nascita:

```
select Nome, Datanascita, TO_CHAR(Datanascita,
      '"Bambina nata il" ddth, fmMonth YYYY "alle" HH.MI "del mattino "')
      "Formattato"
     from COMPLEANNO
    where Nome = 'VITTORIA';
```

NOME	DATANASCITA Formattato
VICTORIA	20-MAY-49 Bambina nata il 20 May 1949, alle 3.27 del mattino

Dopo aver visto questo risultato, si supponga di voler visualizzare la data in lettere. A tal fine occorre utilizzare il controllo **sp**:

```
select Nome, Datanascita, TO_CHAR(Datanascita,  
'"Bambina nata il" Ddsp "di" fmMonth, YYYY, "alle" HH.MI')  
      Formattato  
  from COMPLEANNO  
 where Nome = 'VICTORIA';
```

NOME	DATANASCITA FORMATTATA
VICTORIA	20-MAY-49 Bambina nata il Twenty di May, 1949, alle 3.27

Ora, la data 20 è visualizzata in lettere, ma è ancora errata. Si aggiunga il suffisso **th** a **sp**:

```
select Nome, Datanascita, TO_CHAR(Datanascita,  
'"Bambina nata il" Ddsptn "di" fmMonth, YYYY, "alle" HH.MI')  
      Formattato  
  from COMPLEANNO  
 where Nome = 'VICTORIA';
```

NOME	DATANASCITA FORMATTATO
VICTORIA	20-MAY-49 Bambina nata il Twentieth di May, 1949, alle 3.27

Ma erano le 3.27 del mattino o del pomeriggio? Questa informazione potrebbe essere aggiunta tra virgolette, ma in questo modo il risultato riporterebbe sempre “del mattino” o “del pomeriggio”, indipendentemente dall’ora effettiva del giorno (dal momento che le virgolette racchiudono un letterale). Al contrario, Oracle consente di aggiungere dopo l’ora “del mattino” o “del pomeriggio”, ma non tra virgolette. Quindi, Oracle la interpreta come una richiesta di visualizzazione di questi codici. Nell’esempio seguente si può notare come questo controllo di formattazione sia stato inserito in **select** come P.M., ma il risultato mostra A.M., perché la nascita si è verificata al mattino:

```
select Nome, Datanascita, TO_CHAR(Datanascita,  
'"Bambina nata il" Ddsptn "di" fmMonth, YYYY, "alle" HH.MI P.M.')  
      Formattato  from COMPLEANNO  
 where Nome = 'VICTORIA';
```

NOME	DATANASCITA FORMATTATO
VICTORIA	20-MAY-49 Bambina nata il Twentieth di May, 1949, alle 3.27 A.M.

Un elenco di tutte le possibili opzioni di data è riportato nel riquadro “Formati delle date”, più avanti nel capitolo. Come si fa a creare un formato di data per l’anno 776 dal regno di Luigi IX? Occorre utilizzare l’aritmetica delle date per alterare l’anno da A.D. a A.L. (il regno di Luigi

IX è iniziato nel 1226, perciò si deve sottrarre 1226 dall'anno corrente) e infine formattare semplicemente il risultato utilizzando TO\_CHAR.

## L'errore più comune in TO\_CHAR

È opportuno controllare sempre i formati delle date quando si utilizza TO\_CHAR. L'errore più comune è quello di sostituire il formato MM (mese) a quello MI (minuti) quando si formatta la parte della data che indica l'ora.

Per esempio, per visualizzare l'ora corrente occorre utilizzare la funzione TO\_CHAR effettuando una query in modo da ottenere l'ora da SysDate:

```
select TO_CHAR(SysDate,'HH:MI:SS') Ora from DUAL;
```

```
Ora
-----
10:01:30
```

Questo esempio è corretto, dal momento che utilizza MI per mostrare i minuti. Tuttavia, gli utenti selezionano spesso MM, forse perché stanno impiegando altre due coppie di lettere doppie, HH e SS. Se si seleziona MM, viene restituito il mese, non i minuti:

```
select TO_CHAR(SysDate,'HH:MM:SS') OraNo from DUAL;
```

```
OraNo
-----
10:03:30
```

In questo caso, l'ora non è corretta, perché al posto dei minuti è stato selezionato il mese. Anche se Oracle è flessibile e supporta molti formati differenti per le date, non può evitare che gli utenti commettano questo errore.

## NEW\_TIME: passaggio tra i vari fusi orari

La funzione NEW\_TIME indica l'ora e la data di una colonna di date o di date letterali con altri fusi orari. Questa è la sintassi per NEW\_TIME:

```
NEW_TIME(data,'questo','quello')
```

*data* è la data (e l'ora) nel fuso orario *questo*. *questo* dev'essere sostituito da un'abbreviazione di tre lettere per il fuso orario corrente, mentre *quello* dev'essere sostituito da un'abbreviazione di tre lettere per il fuso orario di cui si vuole conoscere la data e l'ora. Le opzioni per i fusi orari sono fornite nel paragrafo "Funzioni di data", riportato in precedenza in questo capitolo. Per confrontare la data della nascita di Vittoria tra l'ora standard dell'Est e quella delle Hawaii, senza mostrare l'ora, occorre utilizzare l'istruzione seguente:

```
select Datanascita, NEW_TIME(Datanascita,'EST','HST')
  from COMPLEANNO
 where Nome = 'VICTORIA';
```

```
DATANASCITA NEW_TIME(
-----
20-MAY-49 19-MAY-49
```

Come è possibile che Vittoria sia nata in due giorni diversi? Dato che ogni data in Oracle contiene anche l'ora, è sufficiente utilizzare TO\_CHAR e NEW\_TIME per scoprire le differenze di data e di ora tra i due fusi orari. Quanto segue risponde alla domanda:

```
select TO_CHAR(Datanascita,'fmMonth Ddth, YYYY "alle" HH:MI AM') Nata,
       TO_CHAR(NEW_TIME(Datanascita,'EST','HST'),
               'fmMonth ddth, YYYY "alle" HH:MI AM') Nata
  from COMPLEANNO
 where Nome = 'VICTORIA';
```

NATA	NATA
-----	-----
May 20th, 1949 alle 3.27 AM	May 19th, 1949 alle 10.27 PM

## Calcoli di TO\_DATE

TO\_DATE segue le stesse convenzioni di formattazione di TO\_CHAR, con alcune restrizioni. Lo scopo di questa funzione è convertire una stringa letterale, come MAY 20, 1949, in un formato di data di Oracle. In questo modo, la data potrà essere impiegata nei calcoli con le date.

Ecco la sintassi di TO\_DATE:

```
TO_DATE(stringa[, 'formato'])
```

Per convertire la stringa 22-FEB-02 in un formato di data di Oracle, si utilizza quanto segue:

```
select TO_DATE('22-FEB-02', 'DD-MON-YY') from DUAL;
```

```
TO_DATE(
-----
22-FEB-02
```

Occorre osservare, però, che il formato 22-FEB-02 è già quello predefinito in cui Oracle visualizza e accetta le date. Quando una stringa letterale ha una data in questo formato, *formato* in TO\_DATE può essere tralasciato, ottenendo esattamente lo stesso risultato:

```
select TO_DATE('22-FEB-02') from DUAL;
```

```
TO_DATE(
-----
22-FEB-02
```

Ma in quale secolo cade la data? È il 1902 o il 2002? Se non si specifica il valore completo di quattro cifre per l'anno, allora il valore appropriato del secolo dipende dalle impostazioni predefinite del database.

Se la stringa è in un formato simile, ma non proprio quello predefinito di Oracle, ovvero DD-MON-YY, TO\_DATE fallisce:

```
select TO_DATE('02/22/02') from DUAL;
```

```
ERROR: ORA-01843: not a valid month
```

Quando il formato (*formato*) corrisponde alla stringa letterale (*stringa*), questa viene convertita in data e visualizzata nel formato di data predefinito:

---

```
select TO_DATE('02/22/02','MM/DD/YY') from DUAL;
TO_DATE(
-----
22-FEB-02
```

Si supponga che sia necessario conoscere il giorno della settimana del 22 febbraio. La funzione TO\_CHAR non funziona, anche con la stringa letterale nel formato appropriato, perché richiede una data (si veda la sintassi all'inizio del paragrafo “Formattazione di TO\_DATE e TO\_CHAR”):

```
select TO_CHAR('22-FEB-02','Giorno') from DUAL;
ORA-01722: invalid number
```

Il messaggio è un po' fuorviante, ma il punto è che la query fallisce. Si potrebbe ricorrere alla funzione EXTRACT, ma perché questa query funzioni, occorre prima convertire la stringa in una data. Quindi occorre combinare due funzioni TO\_CHAR e TO\_DATE:

```
select TO_CHAR( TO_DATE('22-FEB-02'),'Giorno') from DUAL;
TO_CHAR(TO_DATE('22-FEB-02'),'GIORNO')
-----
Friday
```

TO\_DATE può accettare anche dei numeri, senza gli apici, invece di stringhe, purché siano formattati in modo coerente. Ecco un esempio:

```
select TO_DATE(11051946,'MMDDYYYY') from DUAL;
TO_DATE(1
-----
05-NOV-46
```

I segni di interruzione in *formato* vengono ignorati, ma il numero deve seguire l'ordine dei controlli di formato. Il numero stesso non deve contenere segni di interruzione.

Quanto può essere complesso il controllo di formato in TO\_DATE? Si supponga di aver semplicemente invertito l'istruzione select con TO\_CHAR mostrata in precedenza, mettendo il risultato in una parte *stringa* di TO\_DATE e mantenendo lo stesso formato di TO\_CHAR.

```
select TO_DATE('Bambina nata il Twentieth di May, 1949, alle 3:27 A.M.',
  '"Bambina nata il" Ddsptn "di" fmMonth, YYYY, "alle" HH:MI P.M.')
  Formattato
  from COMPLEANNO
  where Nome = 'VICTORIA';

ERROR: ORA-01858: a non-numeric character was found
      where a numeric was expected
```

Si è verificato chiaramente un errore. Come si è visto, può essere utilizzato solo un numero limitato di controlli di *formato*.

Queste sono le restrizioni sul *formato* che controlla TO\_DATE:

- Non sono consentite stringhe letterali, come “Bambina nata il”.
- I giorni non possono essere scritti in lettere. Devono essere dei numeri.

- Sono consentiti i segni di interpunkzione.
- fm non è necessario. Se utilizzato, viene ignorato.
- Se viene utilizzato Month, il mese nella stringa dev'essere scritto in lettere. Se viene utilizzato Mon, il mese dev'essere un'abbreviazione di tre lettere. Le lettere maiuscole e minuscole vengono ignorate.

Questa istruzione select funziona:

```
select TO_DATE('May 20, 1949, 3:27 A.M. ', 'Month Dd,
               YYYY, HH:MI P.M.')
      Formattato
   from COMPLEANNO
  where Nome = 'VICTORIA';
```

FORMATTO

-----

20-AUG-49

## 9.4 Date nelle clausole where

All'inizio di questo capitolo si è visto un esempio di aritmetica delle date utilizzata in una clausola **where**:

```
select Festa, Dataeffettiva, DataCelebrata
      from Festa
     where DataCelebrata - Dataeffettiva != 0;
```

FESTA	DATAEFF	DATACELEBRATA
MARTIN LUTHER KING, JR.	15-JAN-02	17-JAN-02
LINCOLNS BIRTHDAY	12-FEB-02	18-FEB-02
WASHINGTONS BIRTHDAY	22-FEB-02	18-FEB-02
MEMORIAL DAY	30-MAY-02	27-MAY-02
COLUMBUS DAY	12-OCT-02	14-OCT-02

Le date possono essere impiegate anche con altri operatori logici di Oracle, fermi restando alcuni avvertimenti e restrizioni. L'operatore BETWEEN applica l'aritmetica delle date se la colonna che lo precede è una data, anche se le date di confronto sono stringhe letterali:

```
select Festa, DataCelebrata
      from FESTA
     where DataCelebrata BETWEEN
           '01-JAN-02' and
           '22.02.02';
```

FESTA	DATACELEBRATA
NEW YEAR DAY	01-JAN-02
MARTIN LUTHER KING, JR.	17-JAN-02
LINCOLNS BIRTHDAY	18-FEB-02
WASHINGTONS BIRTHDAY	18-FEB-02
FAST DAY, NEW HAMPSHIRE	22-FEB-02

L'operatore logico IN funziona anche con le stringhe letterali:

```
select Festa, DataCelebrata
  from FESTA
 where DataCelebrata IN ('01-JAN-02', '22-FEB-02');
```

FESTA	DATACELEBRATA
NEW YEAR DAY	01-JAN-02
FAST DAY, NEW HAMPSHIRE	22-FEB-02

Se non si è sicuri che il 2000 sia il secolo predefinito, si può utilizzare la funzione TO\_DATE per specificare i valori del secolo per le date nell'operatore IN:

```
select Festa, DataCelebrata
  from FESTA
 where DataCelebrata IN
       (TO_DATE('01-JAN-2002', 'DD-MON-YYYY'),
        TO_DATE('22-FEB-2002', 'DD-MON-YYYY'));
```

FESTA	DATACELEBRATA
NEW YEAR DAY	01-JAN-02
FAST DAY, NEW HAMPSHIRE	22-FEB-02

LEAST e GREATEST non funzionano, perché presuppongono che le stringhe letterali siano *stringhe*, non *date*. Per una spiegazione esaurente di LEAST e GREATEST, fare riferimento al riquadro “Un avvertimento su GREATEST e LEAST” più indietro nel capitolo.

## 9.5 Gestione di più secoli

Se nelle applicazioni si utilizzano solo valori di due cifre per gli anni, si possono avere problemi per l'anno 2000. Se si indicano solo due cifre per un anno (come '98' per '1998'), allora è compito del database specificare il valore del secolo ('19') quando il record viene inserito. Se si inseriscono delle date precedenti all'anno 2000 (per esempio, date di nascita), si potrebbero avere dei problemi con i valori del secolo assegnati alla data.

In Oracle, tutti i valori di date hanno il secolo. Se si specificano soltanto le ultime due cifre dell'anno, per default Oracle utilizza il secolo corrente come valore del secolo quando inserisce un record. Per esempio, il listato seguente mostra un inserimento nella tabella COMPLEANNO.

```
insert into COMPLEANNO
(Nome, Cognome, Datnascita.)
values
('ALICIA', 'ANN', '21-NOV-39');
```

Nell'esempio precedente non è specificato alcun valore per il secolo nella colonna Datnascita e nessuna età. Se si utilizza la funzione TO\_CHAR nella colonna Datnascita, si può vedere la data di nascita completa inserita da Oracle, impostata per default al secolo corrente:

```
select TO_CHAR(Datnascita, 'DD-MON-YYYY') Nata
  from COMPLEANNO
```

```
where Nome='ALICIA'  
and Cognome='ANN';
```

NATA

-----  
21-NOV-2039

Per le date che possono essere impostate correttamente per default al secolo corrente, l'uso del valore predefinito non rappresenta un problema. Il valore della Datanascita di Alicia è il 21-NOV-2039, una data errata di ben 100 anni! Ogni volta che si inseriscono i valori delle date, sarebbe opportuno specificare il valore completo dell'anno con quattro cifre.

## 9.6 Uso della funzione EXTRACT

A partire da Oracle9*i*, al posto della funzione TO\_CHAR è possibile utilizzare la funzione EXTRACT quando si selezionano parti dei valori di data, per esempio il mese o il giorno. La sintassi per la funzione EXTRACT è la seguente:

```
EXTRACT  
( { { YEAR  
| MONTH  
| DAY  
| HOUR  
| MINUTE  
| SECOND  
}  
| { TIMEZONE_HOUR  
| TIMEZONE_MINUTE  
}  
| { TIMEZONE_REGION  
| TIMEZONE_ABBR  
}  
)  
FROM { dataora_valore_espressione | intervallo_valore_espressione }
```

Per esempio, se si desiderasse estrarre il mese in cui è nata Vittoria, si potrebbe eseguire la query seguente:

```
select Datanascita, EXTRACT(Mese from COMPLEANNO) Mese  
from COMPLEANNO  
where Nome = 'VICTORIA';
```

DATANASCITA	MESE
20-MAY-49	5

Per operazioni di estrazioni più complesse, si dovrà utilizzare TO\_CHAR, tuttavia EXTRACT è in grado di supportare molte query comuni relative a valori di date.

## 9.7 Uso dei tipi di dati TIMESTAMP

A partire da Oracle9i, è possibile sfruttare i vantaggi offerti da numerosi tipi di dati nuovi. Il tipo di dati DATE memorizza la data e l'ora precisa al secondo; i tipi di dati TIMESTAMP invece memorizzano la data includendo anche i miliardesimi di secondo.

Il tipo di dati di base per i valori dell'indicatore orario prende il nome di TIMESTAMP. Come DATE, anch'esso memorizza l'anno, il mese, il giorno, l'ora, i minuti e i secondi. Inoltre, include un'impostazione *precisione alla frazione di secondo*, che determina il numero di cifre nella parte frazionaria del campo dei secondi. Per default, la precisione è 6; quindi i valori validi sono quelli compresi tra 0 e 9.

Nell'esempio seguente, viene creata una tabella con il tipo di dati TIMESTAMP, e viene popolata tramite la funzione SYSTIMESTAMP:

```
create table X1
(tscol  TIMESTAMP(5));

insert into X1 values (SYSTIMESTAMP);
```

Ora, dalla tabella viene selezionato questo valore:

```
select * from X1;

TSCOL
-----
15-MAR-02 04.58.06.00542 PM
```

L'output mostra il secondo in cui è stata inserita la riga, addirittura fino a cinque posizioni dopo il valore decimale.

La funzione SYSTIMESTAMP restituisce i dati sotto forma di TIMESTAMP (*precisione alla frazione di secondo*) con il tipo di dati WITH TIME ZONE. La stessa identica riga, inserita in una colonna definita con il tipo di dati TIMESTAMP(5) WITH TIME ZONE, restituisce i dati nel formato seguente:

```
create table X2
(tscol  TIMESTAMP(5) WITH TIME ZONE);

select * from X2;

TSCOL
-----
15-MAR-02 04.58.06.00542 PM -05:00
```

In questo output, il fuso orario viene visualizzato come una differenza di Coordinated Universal Time (UTC). Attualmente, il database è impostato a un fuso orario che è cinque ore avanti rispetto al valore di UTC.

Oracle supporta anche un tipo di dati TIMESTAMP (*precisione alla frazione di secondo*) WITH LOCAL TIME ZONE, simile a TIMESTAMP WITH TIME ZONE. L'unica differenza è che i dati vengono normalizzati con il fuso orario del database quando sono memorizzati nel database, quindi durante le operazioni di recupero gli utenti visualizzano i dati nel fuso orario di sessione.

Oltre ai tipi di dati TIMESTAMP, Oracle supporta anche due tipi di dati di intervallo: INTERVAL YEAR (*precisione all'anno*) TO MONTH e INTERVAL DAY (*precisione al giorno*) TO SECOND (*precisione alla frazione di secondo*). INTERVAL YEAR TO MONTH memorizza un lasso di tempo espresso in anni e mesi, dove la precisione è il numero di cifre contenute nel campo YEAR (compreso tra 0 a 9; il valore predefinito è di 2 cifre). Il tipo di dati INTERVAL DAY TO SECOND memorizza un lasso di tempo espresso in giorni, ore, minuti e secondi; la precisione per i valori del giorno e dei secondi accetta numeri compresi tra 0 e 9. I tipi di dati INTERVAL vengono utilizzati principalmente nelle analisi statistiche e nelle procedure di data mining.

## Capitolo 10

# Funzioni di conversione e trasformazione

- 10.1 **Funzioni di conversione elementari**
- 10.2 **Funzioni di conversione specializzate**
- 10.3 **Funzioni di trasformazione**
- 10.4 **Conclusioni**

In questo capitolo vengono esaminate le funzioni che convertono un tipo di dati in un altro. Finora sono stati trattati esaurientemente quattro tipi di dati principali e le loro funzioni associate:

- CHAR (stringhe di caratteri a lunghezza fissa) e VARCHAR2 (stringhe di caratteri a lunghezza variabile) includono qualsiasi lettera dell'alfabeto, qualsiasi numero o qualsiasi simbolo sulla tastiera. I letterali carattere devono essere racchiusi tra apici: 'Salute a tutti!'.
- NUMBER include solo le cifre da 0 a 9, un punto decimale e un segno meno, se necessario. I letterali NUMBER non sono racchiusi fra apici: -246.320
- DATE è un tipo speciale che include informazioni sulla data, l'ora e il fuso orario. Ha un formato predefinito, DD-MON-YY, ma può essere visualizzato in molti modi utilizzando la funzione TO\_CHAR, come si è mostrato nel Capitolo 9. I letterali DATE devono essere racchiusi fra apici: '26-AUG-81'.

Ognuno di questi tipi di dati ha un gruppo di funzioni progettate appositamente per la loro manipolazione, come si è visto nei Capitoli 7, 8 e 9. Le funzioni di stringa sono utilizzate con colonne o letterali di caratteri, mentre le funzioni aritmetiche con colonne o letterali NUMBER, infine le funzioni di data con colonne o letterali DATE. La maggior parte delle funzioni di gruppo e miste opera con qualsiasi tipo di dati. Alcune di queste funzioni modificano l'oggetto su cui agiscono (CHAR, VARCHAR2, NUMBER o DATE), mentre altre riportano informazioni sull'oggetto.

In un certo senso, molte delle funzioni esaminate finora sono *funzioni di trasformazione*, ossia modificano gli oggetti. Tuttavia, quelle trattate in questo capitolo cambiano gli oggetti in modo inusuale: li trasformano da un tipo di dati a un altro o effettuano una profonda trasformazione dei dati. La Tabella 10.1 descrive queste funzioni.

L'uso di due di queste funzioni, TO\_CHAR e TO\_DATE, è già stato esaminato nel Capitolo 9. TO\_CHAR trasforma una data in una stringa di caratteri (nel formato richiesto). TO\_CHAR è in grado di convertire non solo date ma anche numeri in stringhe di caratteri. Anche TO\_DATE è una funzione di trasformazione. Prende una stringa di caratteri o un numero e li converte nel tipo di dati DATE. Infine, può essere utilizzata nell'aritmetica delle date per calcolare MONTHS\_BETWEEN, NEXT\_DAY e altre funzioni di data.

**Tabella 10.1** Funzioni di trasformazione.

NOME FUNZIONE	DEFINIZIONE
ASCIISTR	Traduce una stringa in qualunque set di caratteri e restituisce una stringa ASCII nel set di caratteri del database.
BIN_TO_NUM	Converte un valore binario nel suo equivalente numerico.
CAST	Effettua una conversione CAST di un tipo incorporato o collection in un altro; è utilizzata comunemente con tabelle annidate e array variabili.
CHARTOROWID	Modifica una stringa di caratteri in modo che funzioni come un identificatore di riga interno di Oracle o RowID.
COMPOSE	Traduce una stringa in qualunque tipo di dato in una stringa UNICODE nella sua forma completamente normalizzata utilizzando il medesimo set di caratteri della stringa di input.
CONVERT	Converte una stringa di caratteri dal set di caratteri di una lingua in un altro.
DECODE	Decodifica una stringa di caratteri CHAR o VARCHAR2 o un numero (NUMBER) in una serie di vari numeri, in base a un valore. Si tratta di una funzione di tipo <i>if, then, else</i> molto potente. A essa è dedicato il Capitolo 17.
DECOMPOSE	Traduce una stringa in qualunque tipo di dato in una stringa UNICODE dopo la sua scomposizione canonica utilizzando il medesimo set di caratteri della stringa di input.
HEXTORAW	Modifica una stringa di caratteri di numeri esadecimali in binario.
NUMTODSINTERVAL	Converte un numero in un letterale INTERVAL DAY TO SECOND.
NUMTOYMINTERVAL	Converte un numero in un letterale INTERVAL YEAR TO MONTH.
RAWTOHEX	Modifica una stringa di numeri binari in una stringa di caratteri di numeri esadecimali.
RAWTONHEX	Converte un dato RAW in un valore NVARCHAR2 contenente l'equivalente esadecimale.
ROWIDTOCHAR	Modifica un identificatore di riga interno di Oracle, o RowID, in una stringa di caratteri.
ROWIDTONCHAR	Converte un valore RowID in tipo di dati NVARCHAR2.
TO_CHAR	Converte un NUMBER o DATE in una stringa di caratteri.
TO_CLOB	Converte i valori NCLOB in una colonna LOB o in altre stringhe di caratteri in valori CLOB.
TO_DATE	Converte un elemento di tipo NUMBER, CHAR o VARCHAR2 in un elemento di tipo DATE (un tipo di dato di Oracle).
TO_DSINTERVAL	Converte una stringa di tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un tipo INTERVAL DAY TO SECOND.
TO_LOB	Converte un valore LONG in un LOB come parte di un'istruzione <i>insert as select</i> .
TO_MULTI_BYTE	Converte i caratteri a singolo byte di una stringa di caratteri in caratteri di più byte.
TO_NCHAR	Converte una stringa di caratteri, i numeri o le date dal set di caratteri del database in quello della lingua nazionale.
TO_NCLOB	Converte i valori CLOB in una colonna LOB o in altre stringhe di caratteri in valori NCLOB.

(segue)

**Tabella 10.1** Funzioni di trasformazione. (*continua*)

NOME FUNZIONE	DEFINIZIONE
TO_NUMBER	Converte un CHAR o un VARCHAR2 in un NUMBER.
TO_SINGLE_BYTE	Converte i caratteri a più byte di un CHAR o un VARCHAR2 in byte singoli.
TO_YMINTERVAL	Converte una stringa di tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un tipo INTERVAL YEAR TO MONTH .
TRANSLATE	Traduce i caratteri contenuti in una stringa in caratteri differenti.
UNISTR	Converte una stringa in Unicode nel set di caratteri Unicode del database.

## 10.1 Funzioni di conversione elementari

Anche se nella Tabella 10.1 sono elencate numerose funzioni di conversione, le più utilizzate sono tre e il loro scopo è convertire un tipo di dati in un altro:

- TO\_CHAR trasforma un valore DATE o NUMBER in una stringa di caratteri.
- TO\_DATE trasforma un valore NUMBER, un CHAR o un VARCHAR2 in un DATE.
- TO\_NUMBER trasforma un valore CHAR o un VARCHAR2 in un NUMBER.

Qual è l'importanza di queste trasformazioni? TO\_DATE ovviamente è necessaria per applicare l'aritmetica delle date. TO\_CHAR consente di manipolare un numero come se fosse una stringa, utilizzando le funzioni di stringa. TO\_NUMBER permette di usare una stringa che contiene solo numeri come se fosse un numero; con essa è possibile effettuare la somma, la sottrazione, la moltiplicazione, la divisione e così via.

Questo significa che, se si è memorizzato un codice di avviamento postale di nove cifre come un numero, lo si potrebbe trasformare in una stringa e poi utilizzare SUBSTR e la concatenazione per aggiungere un trattino (come quando si stampano gli indirizzi sulle buste):

```
select SUBSTR(TO_CHAR(948033515),1,5)|| '-' ||
       SUBSTR(TO_CHAR(948033515),6) Cap
  from DUAL;
CAP
-----
94803-3515
```

In questo caso, la funzione TO\_CHAR trasforma il numero puro 948033515 (si noti che non è racchiuso tra apici, come avrebbe se si trattasse di una stringa CHAR o VARCHAR2) in una stringa di caratteri. Successivamente, SUBSTR taglia le posizioni dalla 1 alla 5 di questa “stringa”, producendo 94803. All'estremità destra della stringa viene concatenato un trattino, quindi un'altra funzione TO\_CHAR crea un'altra “stringa”, che un'altra SUBSTR taglia dalla posizione 6 fino alla fine. La seconda stringa, 3515, viene concatenata dopo il trattino. L'intera stringa ricostruita viene etichettata come “CAP”, e Oracle la visualizza: 94803-3515. La funzione TO\_CHAR consente di utilizzare le funzioni di manipolazione delle stringhe su numeri (e date) come se fossero effettivamente stringhe. Comodo? Certamente. Tuttavia, si consideri quanto segue:

```
select SUBSTR(948033515,1,5)||'-'||  
      SUBSTR(948033515,6) Cap  
  from DUAL;
```

CAP

```
-----  
94803-3515
```

Ci si aspetterebbe un errore, poiché 948033515 è un numero e non una stringa di caratteri. Però la funzione di stringa SUBSTR funziona comunque. E funzionerebbe con una colonna di database di tipo NUMBER? Ecco una tabella con una colonna Cap definita come NUMBER:

```
describe INDIRIZZO
```

Nome	Null?	Type
COGNOME		VARCHAR2(25)
NOME		VARCHAR2(25)
VIA		VARCHAR2(50)
CITTÀ		VARCHAR2(25)
PROV		CHAR(2)
CAP		NUMBER
TELEFONO		VARCHAR2(12)
PRO		VARCHAR2(5)

Ora viene selezionato solo il codice di avviamento postale per tutte le Mary nella tabella:

```
select SUBSTR(Cap,1,5)||'-'||  
      SUBSTR(Cap,6) Cap  
  from INDIRIZZO  
 where Nome = 'MARY';
```

CAP

```
-----  
94941-4302  
60126-2460
```

SUBSTR funziona proprio come con le stringhe, anche se il codice di avviamento postale è una colonna di tipo NUMBER della tabella INDIRIZZO. Funzionano anche le altre funzioni di stringa?

```
select Cap, RTRIM(Cap,20)  
  from INDIRIZZO  
 where Nome = 'MARY';
```

```
CAP RTRIM(CAP,20)
```

```
-----  
949414302 9494143  
601262460 60126246
```

La colonna a sinistra dimostra che il codice di avviamento postale è un numero; giustificato a destra, come avviene per default per i numeri. La colonna RTRIM invece è giustificata a sinistra, proprio come le stringhe, e gli zero e i due sono stati rimossi dal lato destro dei codici di

avviamento postale. C'è qualcos'altro di particolare. La sintassi di RTRIM, richiamata dal Capitolo 7, è la seguente:

```
RTRIM(stringa [, 'set'])
```

Il *set* di caratteri da rimuovere dalla stringa è racchiuso fra apici, ma in questo esempio:

```
RTRIM(Cap.20)
```

non ci sono apici. Come mai funziona?

## Conversione automatica dei tipi di dati

Oracle converte automaticamente questi numeri, sia il CAP sia il 20, in stringhe, come se fossero entrambi preceduti dalle funzioni TO\_CHAR. In effetti, con alcune evidenti eccezioni, Oracle trasforma automaticamente qualsiasi tipo di dati in un altro in base alla funzione che gli viene applicata. Se si tratta di una funzione di stringa, Oracle converte immediatamente un NUMBER o un DATE in una stringa e la funzione opera correttamente. Se si ha una funzione DATE e la colonna o il letterale è una stringa nel formato DD-MON-YY, Oracle la converte in un DATE. Se si ha una funzione aritmetica e la colonna o il letterale è una stringa di caratteri, Oracle la converte in un NUMBER ed effettua il calcolo.

Il meccanismo funziona sempre? No. Per ottenere una conversione automatica di un tipo di dati in un altro, il primo deve “assomigliare” a quello in cui deve essere convertito. Ecco alcune linee guida fondamentali:

- Qualsiasi NUMBER o DATE può essere convertito in una stringa di caratteri. Qualsiasi funzione di stringa può essere utilizzata su una colonna di tipo NUMBER o DATE. I letterali NUMBER non devono essere racchiusi fra apici quando vengono utilizzati in una funzione di stringa, i letterali DATE sì.
- Un valore CHAR o VARCHAR2 viene convertito in un NUMBER se contiene solo numeri, un punto decimale e un segno meno sulla sinistra.
- Un valore CHAR o VARCHAR2 viene convertito in un DATE solo se è nel formato di data predefinito (in genere DD-MON-YY). Questo vale per tutte le funzioni tranne GREATEST e LEAST, che considerano il valore come una stringa, e vale per BETWEEN solo se la colonna a sinistra dopo la parola BETWEEN è di tipo DATE. In caso contrario dev'essere utilizzata la funzione TO\_DATE, con un formato appropriato.

Queste linee guida potrebbero creare un po' di confusione, quindi sarebbe opportuno favorire l'uso di TO\_DATE e di altre funzioni di conversione per essere certi che i valori vengano trattati in modo appropriato. Gli esempi seguenti dovrebbero aiutare a chiarire le linee guida precedenti. Quelli che seguono sono gli effetti su NUMBER e DATE di varie funzioni di stringa, scelte a caso:

```
select INITCAP(LOWER(SysDate)) from DUAL;
INITCAP(LOWER(SYSDATE))
-----
26-Mar-02
```

Si noti che la funzione INITCAP rende maiuscola la prima lettera di “mar” anche se si trova in mezzo alla stringa “26-mar-02”. Questa è una caratteristica di INITCAP che non è limitata alle date, anche se viene illustrata qui per la prima volta. Si osservi l'esempio seguente:

```
select INITCAP('ecco-un_bel.test.di:punteggiatura;per+initcap')
  from DUAL;

INITCAP('ECCO-UN_BEL.TEST,DI:PUNTEGGIATURA;PE
-----
Ecco-Un_Bel.Test,Di:Punteggiatura;Per+Initcap
```

INITCAP rende maiuscola la prima lettera di ogni parola. Infatti, stabilisce l'inizio di una parola basandosi sul fatto che sia preceduta da qualsiasi carattere diverso da una lettera. Si possono anche tagliare e incollare le date, proprio come se fossero stringhe, utilizzando funzioni di stringa:

```
select SUBSTR(SysDate,4,3) from DUAL;

SUB
---
MAR
```

In questo caso, un valore DATE viene riempito a sinistra con dei 9 in modo da ottenere una lunghezza totale di 20:

```
select LPAD(SysDate,20,'9') from DUAL;

LPAD(SYSDATE,20,'9')
-----
9999999999926-MAR-02
```

LPAD, o qualunque altra funzione di stringa, può essere impiegata anche su NUMBER, letterali (come in questo esempio) o colonne:

```
select LPAD(9,20,0) from DUAL;

LPAD(9,20,0)
-----
00000000000000000009
```

Questi esempi mostrano che le funzioni di stringa trattano i valori NUMBER e DATE come se fossero stringhe di caratteri. Il risultato della funzione (quello che viene visualizzato) è esso stesso una stringa di caratteri. Nel prossimo esempio, una stringa (si notino gli apici) viene trattata come un NUMBER dalla funzione numerica FLOOR:

```
select FLOOR(' -323.78') from DUAL;

FLOOR(' -323.78')
-----
-324
```

Nell'esempio seguente, due stringhe di caratteri letterali sono convertite in DATE per la funzione di data MONTHS\_BETWEEN. Essa funziona solo perché le stringhe letterali sono espresse nel formato di data predefinito, DD-MON-YY:

```
select MONTHS_BETWEEN('16-MAY-02','01-NOV-02') from DUAL;

MONTHS_BETWEEN('16-MAY-02','01-NOV-02')
-----
-5.516129
```

Una delle linee guida afferma che un DATE non viene convertito in un NUMBER. Ma di seguito viene proposto un esempio di addizione e sottrazione con una data. Viene violata la direttiva?

```
select SysDate, SysDate+1, SysDate-1 from DUAL;
```

```
SYSDATE    SYSDATE+1 SYSDATE-1
-----
26-MAR-02  27-MAR-02 25-MAR-02
```

La risposta è no, perché l'addizione e la sottrazione sono operazioni di aritmetica delle date, non di aritmetica normale. L'aritmetica delle date (trattata nel Capitolo 9) funziona solo con l'addizione e la sottrazione e solo con valori DATE. La maggior parte delle funzioni converte automaticamente una stringa di caratteri espressa nel formato di data predefinito in un DATE. Un'ecccezione è questo tentativo di sommare una data con un letterale:

```
select '26-MAR-02'+1 from DUAL;
```

```
ERROR: ORA-01720: invalid number
```

L'aritmetica delle date, anche con tipi di dati DATE reali, funziona solo con l'addizione e la sottrazione. Qualsiasi altra funzione aritmetica applicata a una data fallisce. Le date non vengono convertite in numeri, come illustra questo tentativo di dividere per 2 una data:

```
select SysDate /2 from DUAL;
*
```

```
ERROR at line 1: ORA-00932: inconsistent data types
```

Infine, un NUMBER non viene mai automaticamente convertito in un DATE, perché un numero puro non può essere espresso nel formato predefinito per un data, che è DD-MON-YY:

```
select NEXT_DAY(032602,'VENERDI') from DUAL;
*
```

```
ERROR at line 1: ORA-00932: inconsistent data types
```

Per utilizzare un NUMBER in una funzione di data, occorre applicare la funzione TO\_DATE.

## **Un'avvertenza sulla conversione automatica**

Il problema se sia o meno opportuno consentire a SQL di effettuare la conversione automatica dei tipi di dati è controverso. Da una parte essa semplifica e riduce considerevolmente le funzioni necessarie per far funzionare un'istruzione select. Dall'altra, se l'ipotesi sul contenuto della colonna è sbagliata (per esempio, si suppone che una particolare colonna di caratteri contenga sempre un numero e quindi possa essere utilizzata nei calcoli), una query potrebbe a un certo punto interrompersi, producendo un messaggio di errore e causando una perdita di tempo per cercare di trovare il problema. Inoltre, un'altra persona che leggesse l'istruzione select potrebbe rimanere confusa da funzioni che sembrano inappropriate per operare su caratteri o numeri. L'uso di TO\_NUMBER evidenzia il fatto che un valore numerico è sempre previsto anche se la colonna utilizza il tipo di dati VARCHAR2.

Una semplice regola pratica potrebbe essere quella di utilizzare funzioni in cui il rischio è ridotto, come funzioni di manipolazioni di stringhe su numeri, invece di funzioni aritmetiche su

stringhe. A vantaggio proprio e degli altri utenti, è opportuno porre sempre una nota accanto all'istruzione select per segnalare l'utilizzo della conversione automatica di tipo.

## 10.2 Funzioni di conversione specializzate

Come illustra la Tabella 10.1, Oracle supporta diverse funzioni di conversione specializzate. Se si prevede di utilizzare SQLPLUS e Oracle semplicemente per produrre dei report, probabilmente queste funzioni non saranno mai necessarie. Se invece si decide di utilizzare SQLPLUS per aggiornare il database, sarà necessario costruire applicazioni Oracle; se poi si utilizza il National Language Support, queste informazioni alla fine si riveleranno preziose. Le varie funzioni sono tutte elencate per nome nel Capitolo 42.

**NOTA** *La funzione CAST viene utilizzata con le tabelle annidate e con gli array variabili. Per i dettagli si consulta il Capitolo 31. La funzione DECODE verrà trattata nel Capitolo 17.*

Generalmente, le funzioni di conversione accettano un singolo valore come input e restituiscono un singolo valore convertito come output. Per esempio, la funzione BIN\_TO\_NUM converte valori binari in valori numerici decimali. Il suo valore di input è un elenco delle cifre di un valore binario, separate da virgole e considerate come un'unica stringa di input:

```
select BIN_TO_NUM(1,1,0,1) from DUAL;
```

```
BIN_TO_NUM(1,1,0,1)
```

```
-----  
13
```

```
select BIN_TO_NUM(1,1,1,0) from DUAL;
```

```
BIN_TO_NUM(1,1,1,0)
```

```
-----  
14
```

## 10.3 Funzioni di trasformazione

Anche se in un certo senso qualsiasi funzione che modifica il proprio oggetto può essere considerata una funzione di trasformazione, esistono due funzioni inusuali che possono essere utilizzate in molti modi interessanti per controllare l'output in base all'input, invece di limitarsi a trasformarlo. Queste due funzioni sono TRANSLATE e DECODE.

### TRANSLATE

TRANSLATE è una semplice funzione che effettua una sostituzione ordinata carattere per carattere in una stringa. Ecco la sintassi di TRANSLATE:

```
TRANSLATE(stringa,if,then)
```

**TRANSLATE** esamina ogni carattere in *stringa*, quindi controlla *if* per vedere se il carattere è presente nella stringa. Se lo trova, annota la posizione del carattere in *if* e poi esamina la stessa posizione in *then*. **TRANSLATE** sostituisce il carattere in *stringa* con quello in *then*. Normalmente, la funzione viene scritta su un'unica linea, come mostrato di seguito:

```
select TRANSLATE(7671234,234567890,'BCDEFGHIJ')
  from DUAL;

TRANSLA
-----
GFG1BCD
```

Tuttavia può essere più facile da capire se viene suddivisa in due linee (per SQLPLUS non fa differenza, naturalmente):

```
select TRANSLATE(7671234,234567890,
                  'BCDEFGHIJ')
  from DUAL;

TRANSLA
-----
GFG1BCD
```

Quando **TRANSLATE** vede un 7 in *stringa*, cerca un 7 in *if* e lo traduce nel carattere nella stessa posizione in *then* (una "G" maiuscola). Se il carattere non è presente in *if*, non viene tradotto (si osservi ciò che **TRANSLATE** ha fatto con "1").

**TRANSLATE** è tecnicamente una funzione di stringa, tuttavia, come si può vedere, effettua la conversione automatica dei dati e funziona anche con un insieme di stringhe e numeri. Di seguito è riportato un esempio molto semplice di cifratura di codice, dove ogni lettera dell'alfabeto è traslata di una posizione. Molti anni fa le spie utilizzavano metodi di sostituzione di caratteri analoghi per codificare i messaggi prima di inviarli. Il destinatario effettuava semplicemente il processo inverso. Se si traduce il nome di HAL, il computer parlante nel film *2001: Odissea nello spazio*, facendo scorrere l'alfabeto di un carattere si ottiene quanto segue:

```
select TRANSLATE('HAL','ABCDEFIGHJKLMNOPQRSTUVWXYZ',
                  'BCDEFGHIJKLMNOPQRSTUVWXYZ') Chi
  from DUAL;

CHI
---
IBM
```

## DECODE

Se **TRANSLATE** è una sostituzione carattere per carattere, **DECODE** può essere considerata una sostituzione valore per valore. Per ogni valore in un campo, **DECODE** cerca una corrispondenza in una serie di test *if/then*. **DECODE** è una funzione incredibilmente potente, con un'ampia gamma di aree in cui può essere utile. Il Capitolo 17 è interamente dedicato alla trattazione dell'utilizzo avanzato di **DECODE**.

Ecco la sintassi di questa funzione:

```
DECODE(valore,if1,then1,if2,then2,if3,then3,...,else)
```

Qui sono illustrate soltanto tre combinazioni *if/then*, ma in realtà non vi è alcun limite per esse. Per vedere come opera questa funzione, viene ripresa la tabella GIORNALE introdotta nei primi capitoli:

```
select * from GIORNALE;
```

ARGOMENTO	S	PAGINA
Notizie	A	1
Sport	D	1
Editoriali	A	12
Economia	E	1
Meteo	C	2
Televisione	B	7
Nascite	F	7
Annunci	F	8
Vita moderna	B	1
Fumetti	C	4
Film	B	4
Bridge	B	2
Necrologi	F	6
Salute	F	6

Si supponga di voler cambiare il nome di un paio di argomenti. DECODE controlla ciascun *valore* di Argomento, riga per riga. Se il *valore* che trova è ‘Sport’, allora lo sostituisce con ‘Giochi atletici’; se trova ‘Film’, lo sostituisce con ‘Spettacoli’, se trova qualsiasi altra cosa, utilizza il valore di Argomento.

Nel prossimo esempio viene decodificato il numero di pagina. Se il numero di pagina è 1, viene sostituito con le parole “Prima pagina”. Se il numero di pagina è un qualsiasi altro valore, viene concatenato con la parola ‘Pagina’. Questo mostra che *else* può essere una funzione, un letterale o un’altra colonna.

```
select Argomento, Sezione,
       DECODE(Pagina,'1','Prima pagina','Pagina'||Pagina)
  from GIORNALE;
```

ARGOMENTO	S	DECODE(PAGINA,'1','PRIMAPAGINA','PAGINA'  PAGINA)
Notizie	A	Prima pagina
Sport	D	Prima pagina
Editoriali	A	Pagina 12
Economia	E	Prima pagina
Meteo	C	Pagina 2
Televisione	B	Pagina 7
Nascite	F	Pagina 7
Annunci	F	Pagina 8
Vita moderna	B	Prima pagina
Fumetti	C	Pagina 4
Film	B	Pagina 4
Bridge	B	Pagina 2
Necrologi	F	Pagina 6
Salute	F	Pagina 6

Esistono alcune restrizioni per i tipi di dati che possono essere presenti nell'elenco di *if e then*; se ne occuperà il Capitolo 17.

## 10.4 Conclusioni

La maggior parte delle funzioni in Oracle, anche se è designata per un particolare tipo di dati, come CHAR, VARCHAR2, NUMBER e DATE, in realtà funziona anche con gli altri tipi di dati. Ciò è possibile tramite una conversione automatica di tipo. A parte alcune logiche eccezioni e con la speranza di compatibilità future, questo è possibile finché i dati da convertire appaiono “simili” a quelli in cui devono essere convertiti.

Le funzioni di carattere convertono qualsiasi NUMBER o DATE. Le funzioni numeriche convertono un CHAR o un VARCHAR2, se contiene le cifre da 0 a 9, un punto decimale o un segno meno (-) a sinistra. Tuttavia, le funzioni di NUMBER non convertono i valori DATE. Le funzioni per date convertono le stringhe di caratteri se sono nel formato DD-MON-YY, ma non convertono i numeri.

Esistono infine due funzioni, TRANSLATE e DECODE, che cambiano radicalmente i dati su cui agiscono. TRANSLATE effettua una sostituzione di caratteri in base a qualsiasi configurazione specificata, mentre DECODE esegue una sostituzione di valori per qualsiasi configurazione specificata.



## Capitolo 11

# Raggruppamento di righe

- 11.1 **Uso di group by e having**
- 11.2 **Viste di gruppi**
- 11.3 **La potenza delle viste di gruppi**
- 11.4 **Conclusioni**

**F**inora si è visto come SQL permetta di selezionare (select) righe di informazioni dalle tabelle del database, come le clausole where possano limitare il numero di righe restituite in modo da ottenere solo quelle che soddisfano alcune regole e come le righe restituite possano essere disposte in ordine crescente o decrescente utilizzando order by. Inoltre, si è visto anche come le funzioni per dati NUMBER, DATE e caratteri possano convertire i valori nelle colonne e come le funzioni di gruppo possano fornire informazioni sull'intera serie di righe.

Oltre alle funzioni di gruppo analizzate, esistono anche due clausole di gruppo: having e group by. Sono parallele alle clausole where e order by, con la differenza che operano su gruppi e non su singole righe. Queste clausole possono effettuare analisi molto profonde nei dati.

### 11.1 Uso di group by e having

Se si desidera contare i titoli nella libreria, classificati in base al tipo di libro, si dovrebbe scrivere una query come questa:

```
select NomeCategoria, COUNT(*)  
  from BIBLIOTECA  
group by NomeCategoria;
```

Oracle, risponderebbe con :

NOMECategoria	COUNT(*)
NARRAADULTI	6
SAGGIAADULTI	10
CONSADULTI	6
NARRRAGAZZI	5
SAGGIRAGAZZI	1
ILLUSRAGAZZI	3

Si noti la combinazione del nome di una colonna, NomeCategoria, e di una funzione di gruppo, COUNT, nella clausola select. Questa combinazione è possibile solo perché NomeCategoria è indicata nella clausola group by. Se non fosse così, si otterrebbe nuovamente quell'oscuro messaggio incontrato per la prima volta nel Capitolo 8:

```
SQL> select NomeCategoria, COUNT(*) from BIBLIOTECA;
select NomeCategoria, COUNT(*) from BIBLIOTECA
*
ERROR at line 1:
ORA-00937: not a single-group group function
```

Questo perché le funzioni di gruppo, come **SUM** e **COUNT**, sono progettate per fornire informazioni su un gruppo di righe, non sulle singole righe di una tabella. L'errore viene evitato utilizzando **NomeCategoria** nella clausola **group by**, che induce **COUNT** a contare tutte le righe raggruppate per ogni **NomeCategoria**.

La clausola **having** funziona in modo molto simile a **where**, ma la sua logica è riferita soltanto ai risultati di funzioni di gruppo, invece che a colonne o espressioni per singole righe, le quali possono ancora essere selezionate dalla clausola **where**. Di seguito, le righe dell'esempio precedente vengono ulteriormente ridotte a quelle in cui ci sono più di cinque libri in una categoria:

```
select NomeCategoria, COUNT(*)
  from BIBLIOTECA
 group by NomeCategoria
 having COUNT(*) > 5;
```

NOMECategoria	COUNT(*)
NARRADULTI	6
SAGGIADULTI	10
CONSADULTI	6

Per stabilire una valutazione media per categoria, è possibile utilizzare la funzione **AVG**, come mostrato nel listato seguente:

```
select NomeCategoria, COUNT(*), AVG(Valutazione)
  from BIBLIOTECA
 group by NomeCategoria;
```

NOMECategoria	COUNT(*)	AVG(VALUTAZIONE)
NARRADULTI	6	3.66666667
SAGGIADULTI	10	4.2
CONSADULTI	6	3.16666667
NARRRAGAZZI	5	2.8
SAGGIRAGAZZI	1	3
ILLUSRAGAZZI	3	1

**Valutazione** è una colonna di caratteri di tipo **VARCHAR2**, ma contiene dei valori numerici, quindi Oracle può applicare su di essa delle funzioni numeriche (si consulti il Capitolo 10). Qual è la valutazione media totale?

```
select AVG(Valutazione) from BIBLIOTECA;
AVG(VALUTAZIONE)
-----
3.32258065
```

In questo caso non c'è nessuna clausola **group by**, perché l'intero insieme di righe nella tabella **BIBLIOTECA** viene trattato come gruppo. Ora si può utilizzare questo risultato come

parte di una query più ampia: quali categorie hanno valutazioni medie superiori alla valutazione media di tutti i blocchi?

```
select NomeCategoria, COUNT(*), AVG(Valutazione)
  from BIBLIOTECA
 group by NomeCategoria
 having AVG(Valutazione) >
    (select AVG(Valutazione) from BIBLIOTECA);
```

NOMECategoria	COUNT(*)	AVG(VALUTAZIONE)
NARRADULTI	6	3.66666667
SAGGIADULTI	10	4.2

Analizzando di nuovo i listati precedenti, questo risultato è corretto, perché solo due dei gruppi hanno valori di Valutazione medi superiori alla media totale.

Anche se i risultati sono ordinati in base alla colonna NomeCategoria, lo scopo di group by non è produrre una sequenza desiderata, bensì raccogliere righe “simili”. L’ordine in cui compaiono è un effetto collaterale della modalità di funzionamento di group by; group by non è stata progettata per alterare il criterio di ordinamento.

## Aggiunta di order by

La soluzione per creare un ordine alternativo di visualizzazione è l’aggiunta di una clausola order by dopo la clausola having. Si potrebbe aggiungere quella seguente:

```
order by NomeCategoria desc
```

che invertirebbe l’ordine dell’elenco:

```
select NomeCategoria, COUNT(*)
  from BIBLIOTECA
 group by NomeCategoria
 order by NomeCategoria desc;
```

NOMECategoria	COUNT(*)
ILLUSRAGAZZI	3
SAGGIRAGAZZI	1
NARRRAGAZZI	5
CONSADULTI	6
SAGGIADULTI	10
NARRADULTI	6

oppure si potrebbe usare questa:

```
order by COUNT(*) desc
```

ottenendo

NOMECategoria	COUNT(*)
SAGGIADULTI	10

NARRADULTI	6
CONSADULTI	6
NARRRAGAZZI	5
ILLUSRAGAZZI	3
SAGGIRAGAZZI	1

Anche se è possibile usare l'alias della colonna come parte della clausola `order by`, non lo si potrà utilizzare come parte della clausola `having`. L'assegnazione a `COUNT(*)` di un alias come "Contatore" e il tentativo di usare `having Contatore > 1` come clausola in questa query porteranno a un errore "invalid column name":

```
select NomeCategoria, COUNT(*) Contatore
  from BIBLIOTECA
 group by NomeCategoria
 having Contatore > 1
 order by COUNT(*) desc;
```

```
having Contatore > 1
*
ERROR at line 4:
ORA-00904: invalid column name
```

## Ordine di esecuzione

La query precedente contiene una certa quantità di clausole. Di seguito sono riportate le regole utilizzate da Oracle per eseguire ciascuna di esse e l'ordine in cui viene effettuata l'esecuzione:

1. Vengono scelte le righe in base alla clausola `where`.
2. Queste righe vengono raggruppate in base alla clausola `group by`.
3. Per ogni gruppo vengono calcolati i risultati delle funzioni di gruppo.
4. Vengono scelti ed eliminati i gruppi in base alla clausola `having`.
5. I gruppi vengono ordinati in base ai risultati delle funzioni di gruppo nella clausola `order by`. La clausola `order by` deve utilizzare una funzione di gruppo oppure una colonna specificata nella clausola `group by`.

L'ordine di esecuzione è importante in quanto ha un impatto diretto sulle prestazioni delle query. In generale, più record possono essere eliminati attraverso le clausole `where` e più rapida è l'esecuzione delle query. Questo guadagno di prestazioni è dovuto alla riduzione del numero di righe che devono essere elaborate durante l'operazione `group by`.

Se una query utilizza una clausola `having` per eliminare gruppi, è consigliabile controllare se questa condizione può essere riscritta come una clausola `where`. In molti casi questa riscrittura non è possibile. In genere è realizzabile solo quando la clausola `having` è impiegata per eliminare gruppi basati sulle colonne di raggruppamento.

Per esempio, se si avesse questa query:

```
select NomeCategoria, COUNT(*), AVG(Valutazione)
  from BIBLIOTECA
 where Valutazione > 1
 group by Nomecategoria
 having Nomecategoria like 'A%'
 order by COUNT(*) desc;
```

NOMECategoria	COUNT(*)	AVG(VALUTAZIONE)
SAGGIADULTI	10	4.2
NARRADULTI	6	3.66666667
CONSADULTI	6	3.16666667

l'ordine di esecuzione sarebbe:

1. Eliminare le righe basate su:

where Valutazione > 1

2. Raggruppare le restanti righe in base a:

group by NomeCategoria

3. Per ogni NomeCategoria, calcolare:

COUNT

4. Eliminare i gruppi in base a:

having NomeCategoria like 'A%'

5. Ordinare i gruppi rimasti.

Questa query può essere eseguita più velocemente se i *gruppi* eliminati al passaggio 4 vengono eliminati come *righe* nel passaggio 1, poiché in questo caso devono essere raggruppate meno righe (passaggio 2), devono essere svolti meno calcoli (passaggio 3) e non dev'essere eliminato alcun gruppo (passaggio 4). Ognuno di questi passaggi viene eseguito più rapidamente.

Dato che la condizione having in questo esempio non è basata su una colonna calcolata, può essere facilmente sostituita da una condizione where:

```
select NomeCategoria, COUNT(*), AVG(Valutazione)
  from BIBLIOTECA
 where valutazione > 1
   and NomeCategoria like 'A%'
 group by NomeCategoria
 order by COUNT(*) desc;
```

Nella versione modificata vengono raggruppate meno righe, portando così a miglioramenti nelle prestazioni. Via via che il numero di righe nelle tabelle aumenta, il miglioramento delle prestazioni può aumentare drasticamente dall'eliminazione della prima riga.

## 11.2 Viste di gruppi

Nel Capitolo 3 è stata creata una vista di nome INVASIONE per l'Oracolo di Delfi, che univa le tabelle CLIMA e LOCAZIONE. Questa vista sembrava una tabella a tutti gli effetti, con colonne e righe, ma ognuna delle righe conteneva colonne che in realtà provenivano da due tabelle diverse.

Lo stesso processo di creazione di una vista può essere utilizzato con i gruppi. La differenza è che ogni riga contiene informazioni su un gruppo di righe, una sorta di tabella di totali parziali. Per esempio, si consideri la seguente query di gruppi:

```
select NomeCategoria, COUNT(*)
  from BIBLIOTECA
 group by NomeCategoria;
```

È possibile creare una vista basata su questa query, per poi eseguire la query della vista:

```
create or replace view CATEGORIA_CONTEGGIO as
select NomeCategoria, COUNT(*) Contatore
  from BIBLIOTECA
 group by NomeCategoria;
```

```
desc CATEGORIA_CONTEGGIO
```

Nome	Null?	Type
NOMECATEGORY		VARCHAR2(20)
CONTATORE		NUMBER

```
select * from CATEGORIA_CONTEGGIO;
```

NOMECATEGORY	CONTATORE
NARRADULTI	6
SAGGIADULTI	10
CONSADULTI	6
NARRRAGAZZI	5
SAGGIRAGAZZI	1
ILLUSRAGAZZI	3

**NOTA** *Dato che la colonna COUNT(\*) è una funzione, è necessario assegnarle un alias di colonna (in questo caso Contatore) quando si usa la query come base per una vista.*

## Come rinominare le colonne con alias

Si osservi il nome Contatore nella clausola select. La clausola AS Contatore *rinomina* la colonna che segue. Questi nuovi nomi vengono chiamati *alias*, perché sono utilizzati per nascondere i nomi reali delle colonne sottostanti (che sono complessi in quanto contengono funzioni).

Quando si effettua la query sulla vista, si possono (e si devono) utilizzare i nuovi nomi delle colonne:

```
select NomeCategoria, Contatore from CATEGORIA_CONTEGGIO;
```

“Contatore” viene definito come un *alias di colonna*, un altro nome da utilizzare quando si fa riferimento a una colonna. Nella descrizione della vista e nella query non c’è alcuna prova della funzione di raggruppamento eseguita, solo il nome della colonna Contatore. È come se la vista CATEGORIA\_CONTEGGIO fosse una reale tabella con le righe delle somme mensili. Perché?

Oracle prende automaticamente una singola parola, senza apici, e la utilizza per rinominare la colonna che segue. Durante questa operazione, Oracle trasforma la parola (l’alias) in maiuscolo, a prescindere da come era stata digitata. Si può eseguire una verifica confrontando i nomi delle colonne nei comandi create view e describe. Quando si crea una vista, gli apici non devono mai essere inseriti intorno agli alias delle colonne. Gli alias devono rimanere sempre nelle istruzioni create view senza apici. In questo modo, vengono memorizzati in maiuscolo, cosa

### Alias nella creazione di viste

Internamente, Oracle opera con tutti i nomi delle colonne e delle tabelle in maiuscolo. Questo è il modo in cui i nomi sono salvati nel dizionario di dati e in cui ci si attende che siano riportati. Quando vengono digitati gli alias per creare una vista, non dovrebbero mai essere racchiusi tra apici. Con i doppi apici si induce Oracle a salvarli internamente con caratteri sia maiuscoli che minuscoli. In questo caso, Oracle non sarà mai in grado di trovare queste colonne quando si esegue un'istruzione select, a meno di non racchiudere tra apici i nomi delle colonne durante tutte le query.

Pertanto, quando si creano alias per una vista, non si devono mai utilizzare i doppi apici.

necessaria perché Oracle possa trovarli. Per un avvertimento relativo all'uso degli alias, si rimanda al riquadro “Alias nella creazione di viste”.

Ora si hanno i conteggi di Categoria raccolti in una vista. Si potrebbe creare anche un totale per l'intera libreria, utilizzando CONTEGGIOLIBRI sia come nome per la vista sia come alias della colonna per COUNT(\*):

```
create or replace view CONTEGGIOLIBRI as
select COUNT(*) CONTEGGIOLIBRI
  from BIBLIOTECA
```

View created.

Se si esegue la query sulla vista, si scoprirà che ha un solo record:

```
select CONTEGGIOLIBRI
  from CONTEGGIOLIBRI;
CONTEGGIOLIBRI
-----
31
```

Via via che nuove righe vengono aggiunte ed eseguite sulla tabella BIBLIOTECA, le viste CONTEGGIOLIBRI e CATEGORIA\_CONTEGGIO rifletteranno le modifiche dei conteggi.

### 11.3 La potenza delle viste di gruppi

Ora si vedrà la reale potenza di un database relazionale. Sono state create delle viste che contengono i totali per gruppi: per Categoria e per il gruppo intero. Queste viste ora possono essere *unite*, proprio come si è fatto con le tabelle nel Capitolo 3, per mostrare informazioni che prima non erano evidenti. Per esempio, qual è la percentuale dei libri in ogni categoria?

```
select NomeCategoria Contatore, (Contatore/ContaggioLibri)*100 as Percento
  from CATEGORIA_CONTEGGIO, CONTEGGIOLIBRI
 order by NomeCategoria;
```

NOMECategoria	CONTATORE	PERCENTO
NARRADULTI	6	19.3548387
SAGGIADULTI	10	32.2580645

---

CONSADULTI	6	19.3548387
NARRAGAZZI	5	16.1290323
SAGGIRAGAZZI	1	3.22580645
ILLUSRAGAZZI	3	9.67741935

In questa query, due viste sono elencate nella clausola `from`, ma non sono unite in una clausola `where`. Perché? In questo caso particolare, non è necessaria alcuna clausola `where` perché una delle viste, CONTEGGIOLIBRI, restituisce soltanto una riga (come mostrato nel listato precedente). L'unica riga in CONTEGGIOLIBRI viene unita a ciascuna riga di CATEGORIA\_CONTEGGIO, ottenendo così una riga di output per ogni riga contenuta in CATEGORIA\_CONTEGGIO. Sarebbe stato possibile ottenere gli stessi risultati unendo direttamente la *tabella* BIBLIOTECA alla *vista* CONTEGGIOLIBRI, ma come si può vedere, la query è più complicata e difficile da comprendere, e con l'aumento del numero dei gruppi la query diventa sempre più complessa:

```
select NomeCategoria, COUNT(*),
       (COUNT(*)/MAX(ConteggioLibri))*100 Percento
  from BIBLIOTECA, CONTEGGIOLIBRI
 group by NomeCategoria
order by NomeCategoria;
```

Si osservi il calcolo percentuale:

```
(COUNT(*)/MAX(ConteggioLibri))*100 Percento
```

Dato che questo risultato fa parte di una funzione di raggruppamento, ciascuno dei valori deve essere raggruppato. Pertanto, un tentativo iniziale come questo fallirebbe:

```
(COUNT(*)/ConteggioLibri)*100 Percento
```

in quanto ConteggioLibri non è raggruppata. Considerata la presenza di una sola riga nella vista CONTEGGIOLIBRI, è possibile eseguire una funzione MAX su di essa in modo che venga restituita quest'unica riga, raggruppata a se stessa.

Per creare query in grado di confrontare un raggruppamento di righe con un altro, almeno uno dei raggruppamenti dev'essere una vista oppure una “vista in linea” creata nella clausola `from` della query. Al di là di queste restrizioni tecniche, però, le query eseguite con le viste sono più semplici e facili da capire. Se si confrontano gli ultimi due esempi, la differenza per quel che riguarda la chiarezza è evidente. Infatti, le viste nascondono la complessità.

Per applicare il metodo della vista in linea, si inserisca il testo della vista nella clausola `from` assegnando alle colonne degli alias:

```
select NomeCategoria, Contatore, (Contatore/ConteggioLibri)*100 Percento
  from CATEGORIA_CONTEGGIO,
       (select COUNT(*) ConteggioLibri from BIBLIOTECA)
 order by NomeCategoria;
```

In questo esempio, la vista CONTEGGIOLIBRI è stata rimossa dalla clausola `from`, e sostituita dalla sua query base. In questa query, l'alias di ConteggioLibri viene assegnato al risultato della funzione COUNT(\*) eseguita sulla tabella BIBLIOTECA. Nella query principale, questo alias di ConteggioLibri viene poi utilizzato come parte integrante di un calcolo. Con questo metodo di codifica non è necessario creare la vista CONTEGGIOLIBRI. È opportuno fare molta attenzione quando all'interno della stessa query si utilizzano più livelli di raggruppamento: la creazione di viste in genere aiuta a semplificare la creazione e la gestione del codice.

## order by nelle viste

Da un punto di vista rigorosamente teorico, non esiste alcun motivo per avere una clausola `order by` memorizzata in una vista: si potrebbe eseguire una clausola `order by` quando si effettua la query sulla vista. A partire da Oracle8i, Oracle supporta la clausola `order by` all'interno delle viste, come mostrato di seguito:

```
create view BIBLIOTECA_ORDINATA
as select * from BIBLIOTECA
order by Titolo;
```

Con i dati ordinati nella vista, lo sviluppo dell'applicazione potrebbe risultare più semplice. Per esempio, se il codice passa attraverso un gruppo di record, la possibilità di ordinare prima questi record potrebbe semplificare l'elaborazione e il controllo degli errori. Durante lo sviluppo dell'applicazione, si ha la certezza che i dati vengono sempre restituiti in modo ordinato. La query seguente seleziona i valori di Titolo, utilizzando la pseudocolonna NumeroRiga per limitare l'output a nove record:

```
select Titolo from BIBLIOTECA_ORDINATA
where NumeroRiga 10;
```

TITOLO

```
ANNE OF GREEN GABLES
BOX SOCIALS
CHARLOTTE'S WEB
COMPLETE POEMS OF JOHN KEATS
EITHER/OR
EMMA WHO SAVED MY LIFE
GOOD DOG, CARL
GOSPEL
HARRY POTTER AND THE GOBLET OF FIRE
```

Le viste offrono anche maggiore potenza nell'utilizzo dei tipi di dati carattere, NUMBER e DATE, senza doversi preoccupare di aspetti come i mesi che si presentano in ordine alfabetico.

## La logica della clausola having

Nella clausola `having` la scelta della funzione di gruppo, e della colonna su cui opera, potrebbe non avere alcuna relazione con le colonne o le funzioni di gruppo nella clausola `select`:

```
select NomeCategoria, COUNT(*),
       (COUNT(*)/MAX(ConteggioLibri)*100) Percento
  from BIBLIOTECA, CONTEGGIOLIBRI
 group by NomeCategoria
 having Avg(Valutazione) > 4
 order by NomeCategoria;
```

NOMECategoria	COUNT(*)	PERCENTO
SAGGIADULTI	10	32.2580645

In questo esempio, la clausola `having` ha selezionato solo le categorie (group by ha raccolto tutte le righe in gruppi per NomeCategoria) con una valutazione media maggiore di 4. Tutti gli

altri gruppi sono stati eliminati. Per il gruppo che ha soddisfatto questo criterio è invece stata calcolata la percentuale del conteggio totale.

La clausola `having` è molto efficiente per stabilire quali righe di una tabella contengono valori duplicati in colonne specifiche. Per esempio, se si sta cercando di stabilire un nuovo indice unico su una colonna (o su una serie di colonne) di una tabella e la creazione dell'indice fallisce per problemi di unicità dei dati, è facile determinare quali righe hanno causato il problema.

Innanzitutto occorre selezionare le colonne che dovrebbero essere uniche, seguite da una colonna `COUNT(*)`. Occorre poi raggruppare in base alle colonne che dovrebbero essere uniche e utilizzare la clausola `having` per restituire solo i gruppi dove `COUNT(*)>1`. Quindi, soltanto i record restituiti hanno valori duplicati. La query seguente mostra questo tipo di verifica effettuata per la colonna `NomeAutore` della tabella `AUTORE`:

```
select NomeAutore, COUNT(*)
  from AUTORE
 group by NomeAutore
 having COUNT(*)>1
 order by NomeAutore;
```

Quali libri hanno più di un autore? Si selezionino i titoli da `BIBLIOTECA_AUTORE` per i quali il gruppo (raggruppato per Titolo) ha più di un membro:

```
column Titolo format a40
```

```
select Titolo, COUNT(*)
  from BIBLIOTECA_AUTORE
 group by Titolo
 having COUNT(*)>1;
```

TITOLO	COUNT(*)
COMPLETE POEMS OF JOHN KEATS	2
JOURNALS OF LEWIS AND CLARK	4
KIERKEGAARD ANTHOLOGY	2
RUNAWAY BUNNY	2

Chi sono questi dieci autori? Si potrebbe creare una vista basata su questa query, oppure concepirla come vista in linea:

```
column Titolo format a40
column NomeAutore format a30

select Titolo, NomeAutore
  from BIBLIOTECA_AUTORE,
       (select Titolo TitoloRaggruppato, COUNT(*) ContatoreTitolo
        from BIBLIOTECA_AUTORE
        group by Titolo
        having COUNT(*) > 1)
 where Titolo = TitoloRaggruppato
 order by Titolo, NomeAutore;
```

TITOLO	NOMEAUTORE
COMPLETE POEMS OF JOHN KEATS	JOHN BARNARD
COMPLETE POEMS OF JOHN KEATS	JOHN KEATS
JOURNALS OF LEWIS AND CLARK	BERNARD DE VOTO

JOURNALS OF LEWIS AND CLARK	MERIWETHER LEWIS
JOURNALS OF LEWIS AND CLARK	STEPHEN AMBROSE
JOURNALS OF LEWIS AND CLARK	WILLIAM CLARK
KIERKEGAARD ANTHOLOGY	ROBERT BRETALL
KIERKEGAARD ANTHOLOGY	SOREN KIERKEGAARD
RUNAWAY BUNNY	CLEMENT HURD
RUNAWAY BUNNY	MARGARET WISE BROWN

Questa query potrebbe apparire complicata (e l'uso di una vista la renderebbe più leggibile), tuttavia si basa su concetti analizzati in questo capitolo: una vista in linea esegue una funzione group by e utilizza una clausola having per restituire solamente i titoli con più autori. Questi titoli serviranno poi come base di una query sulla tabella BIBLIOTECA\_AUTORE. In una query singola, la tabella BIBLIOTECA\_AUTORE viene sottoposta a query per i dati raggruppati e per i dati delle singole righe.

### order by con colonne e funzioni di gruppo

La clausola order by viene eseguita dopo le clausole where, group by e having. Può impiegare funzioni di gruppo o colonne di group by, o una combinazione di entrambe. Se utilizza una funzione di gruppo, la funzione effettua le operazioni sui gruppi, poi order by *ordina* i risultati della funzione. Se order by impiega una colonna di group by, ordina le righe restituite in base a questa colonna. Le funzioni di gruppo e le singole colonne (purché siano in group by) possono essere combinate in order by.

Nella clausola order by, si possono specificare una funzione di gruppo e la colonna su cui agisce, anche se non hanno alcuna relazione con le funzioni di gruppo o le colonne presenti in select, group by o having. Se invece in order by si specifica una colonna che non fa parte di una funzione di gruppo, questa colonna dev'essere necessariamente citata nella clausola group by. Si prenda l'ultimo esempio e si modifichi la clausola order by:

```
order by ContatoreTitolo desc, Titolo, NomeAutore
```

I titoli e gli autori verranno ordinati in base al numero degli autori (partendo dal numero maggiore), quindi per Titolo e per NomeAutore:

TITOLO	NOMEAUTORE
JOURNALS OF LEWIS AND CLARK	BERNARD DE VOTO
JOURNALS OF LEWIS AND CLARK	MERIWETHER LEWIS
JOURNALS OF LEWIS AND CLARK	STEPHEN AMBROSE
JOURNALS OF LEWIS AND CLARK	WILLIAM CLARK
COMPLETE POEMS OF JOHN KEATS	JOHN BARNARD
COMPLETE POEMS OF JOHN KEATS	JOHN KEATS
KIERKEGAARD ANTHOLOGY	ROBERT BRETALL
KIERKEGAARD ANTHOLOGY	SOREN KIERKEGAARD
RUNAWAY BUNNY	CLEMENT HURD
RUNAWAY BUNNY	MARGARET WISE BROWN

### Unione di colonne

Come si è spiegato nei Capitoli 1 e 3, per unire due tavelli, è necessario che queste abbiano una relazione definita da una colonna comune. Ciò vale anche se si desidera unire delle viste tra loro, oppure tavelli e viste. L'unica eccezione si ha quando una delle tavelli o delle viste ha una sola

riga, come la tabella CONTEGGIOLIBRI. In questo caso, SQL unisce la singola riga a ogni riga dell'altra tabella o vista e non è necessario fare alcun riferimento, nella clausola where della query, alle colonne di unione.

Qualunque tentativo di unire due tabelle che hanno ognuna più di una riga senza specificare le colonne di unione nella clausola where produce un *prodotto cartesiano*, in genere un risultato gigantesco dove ogni riga di una tabella è unita con ogni riga dell'altra. Una piccola tabella di 80 righe unita in questo modo con un'altra piccola tabella di 100 righe produrrebbe 8000 righe, poche delle quali significative.

## 11.4 Conclusioni

In Oracle, le tabelle possono essere raggruppate in collezioni di righe correlate, come per Titolo, NomeAutore o NomeCategoria. Questo risultato è ottenuto grazie all'uso della clausola group by, che raggruppa solo le righe della tabella che superano il test logico della clausola where:

```
where NomeCategoria like 'A%'  
group by NomeCategoria
```

La clausola having esamina questi gruppi e li elimina in base al risultato del test logico della funzione di gruppo in essa utilizzata, come:

```
having COUNT(*) > 1
```

Vengono restituiti i gruppi per i quali il valore di COUNT(\*) è maggiore di uno. Ciascun gruppo ha solo una riga nella tabella risultante che viene visualizzata. Non è necessario (e in genere non accade) che la clausola having corrisponda alle funzioni di gruppo nella clausola select. Dopo che queste righe sono state selezionate dalla clausola having, devono essere poste nell'ordine desiderato con order by:

```
order by COUNT(*)
```

order by deve utilizzare una colonna menzionata in group by o qualsiasi funzione di gruppo appropriata, che può riferirsi a qualsiasi colonna indipendentemente dalla clausola select o having. La funzione di gruppo effettua i calcoli riga per riga per ogni gruppo creato dalla clausola group by.

Tutte queste potenti caratteristiche di raggruppamento possono essere combinate per creare complesse viste di summary della tabella di base, che appaiono molto semplici. Una volta create, le loro colonne possono essere manipolate e le loro righe selezionate, proprio come avviene per qualunque altra tabella. Queste viste possono anche essere unite fra loro e con tabelle, per produrre analisi profonde nei dati. Le viste in linea possono essere utilizzate per ridurre il numero di oggetti da gestire. All'interno delle viste si possono usare le clausole order by. Per i raggruppamenti comunemente utilizzati, sarebbe opportuno prendere in considerazione l'eventualità di usare le viste per semplificare lo sviluppo dell'applicazione. Nei prossimi capitoli verranno introdotti altri metodi e funzioni che consentono di eseguire manipolazioni di dati ancora più complesse.

## Capitolo 12

# Quando una query dipende da un'altra

- 12.1 **Subquery avanzate**
- 12.2 **Outer join**
- 12.3 **Join naturali e interni**
- 12.4 **UNION, INTERSECT e MINUS**

Questo capitolo e quello successivo introducono concetti più difficili rispetto a quelli esaminati in precedenza. Anche se molte di queste tecniche sono raramente utilizzate nelle normali operazioni di esecuzione di query o di produzione di report, esistono sicuramente occasioni in cui è necessario impiegarle. Se sembrano troppo impegnative, è comunque opportuno andare avanti. Il vantaggio è che quando questi metodi saranno necessari, si sarà in grado di utilizzarli.

### 12.1 Subquery avanzate

Nei capitoli precedenti, si sono già incontrate delle *subquery*, istruzioni select che fanno parte della clausola where di una precedente istruzione select. Le subquery possono essere impiegate anche nelle istruzioni insert, update e delete. Questo aspetto del loro utilizzo verrà trattato nel Capitolo 15.

Spesso una subquery fornisce un approccio alternativo a una query. Per esempio, si supponga di voler sapere quali categorie di libri sono state prese in prestito. Il seguente join triangolare fornisce queste informazioni:

```
select distinct C.CategoriaPadre, C.SottoCategoria
  from CATEGORIA C, BIBLIOTECA B, BIBLIOTECA_PRESTITO BC
 where C.NomeCategoria = B.NomeCategoria
   and B.Titolo = BC.Titolo;
```

CATPADRE SOTTOCATEGORIA

-----	-----
ADULTI	NARRATIVA
ADULTI	SAGGI
ADULTI	CONSULTAZIONE
RAGAZZI	NARRATIVA
RAGAZZI	ILLUSTRATI

Tre tabelle vengono unite tramite join allo stesso modo di due. Le colonne comuni sono impostate uguali fra loro nella clausola where, come mostrato nel listato precedente. Per unire tramite join tre tabelle, occorre collegarne due a una terza. In questo esempio, la tabella CATEGORIA viene unita tramite join alla tabella BIBLIOTECA, e il risultato di questo join viene a

sua volta collegato alla tabella BIBLIOTECA\_PRESTITO. La clausola distinct indica a Oracle di restituire solamente combinazioni distinte di CategoriaPadre e Sottocategoria.

**NOTA** *Non tutte le tabelle vengono unite tramite join. In effetti, il numero di collegamenti tra le tabelle è in genere inferiore di uno rispetto al numero delle tabelle da unire tramite join.*

Una volta unite le tabelle, come mostrato nelle prime due linee della clausola where, è possibile stabilire il numero di verifiche per categoria padre e per sottocategoria.

## Subquery correlate

Esiste un altro modo per eseguire join tra più tabelle? Si ricordi che una clausola where può contenere un'istruzione select di una subquery. Le istruzioni select di subquery possono essere annidate, ovvero una clausola where in una subquery può anche contenere una clausola where con una subquery, che a sua volta può contenere una clausola where con una subquery, fino a molti livelli, più di quanti saranno mai necessari. Il listato seguente mostra tre istruzioni select, ciascuna connessa all'altra attraverso una clausola where:

```
select distinct C.CategoriaPadre, C.SottoCategoria
  from CATEGORIA C
 where NomeCategoria in
       (select NomeCategoria from BIBLIOTECA
        where Titolo in
              (select Titolo from BIBLIOTECA_PRESTITO)
       );

```

CATPADRE SOTTOCATEGORIA

CATPADRE	SOTTOCATEGORIA
ADULTI	NARRATIVA
ADULTI	SAGGI
ADULTI	CONSULTAZIONE
RAGAZZI	NARRATIVA
RAGAZZI	ILLUSTRATI

Questa query seleziona qualsiasi categoria contenga libri che sono stati presi in prestito. Ottiene questo risultato richiedendo semplicemente un libro il cui Titolo si trovi nella tabella BIBLIOTECA e il cui record di controllo sia invece nella tabella BIBLIOTECA\_PRESTITO. In una subquery, Oracle presuppone che le colonne provengano dalla prima istruzione select, ovvero quella che contiene la subquery nella sua clausola where. Questa prende il nome di subquery *annidata*, perché per ogni NomeCategoria nella query principale (esterna), NomeCategoria potrebbe essere correlata nella seconda clausola where.

In altre parole, una subquery può fare riferimento a una colonna di una tabella utilizzata nella propria query principale (la query che ha la subquery nella clausola where). Si consideri la query riportata di seguito:

```
select Titolo from BIBLIOTECA_AUTORE
  where Titolo in
       (select Titolo from BIBLIOTECA
        where NomeAutore = 'STEPHEN JAY GOULD');
```

TITOLO

---

-----  
WONDERFUL LIFE  
THE MISMEASURE OF MAN

Perché questa query funziona? Di per sé, la subquery fallirebbe:

```
select Titolo from BIBLIOTECA
  where NomeAutore = 'STEPHEN JAY GOULD';

where NomeAutore = 'STEPHEN JAY GOULD'
      *
ERROR at line 2:
ORA-00904: invalid column name
```

Quando viene eseguita come subquery, viene correlata alla query padre, quindi è possibile fare riferimento alle colonne della prima istruzione select nella subquery. In questo e nei prossimi capitoli verranno proposti altri esempi di subquery correlate.

## Coordinazione di test logici

Se un lettore cerca un maggior numero di libri in un determinata categoria, quali autori dovrebbe leggere? Si supponga che Fred Fuller, che ha letto due biografie, chieda dei consigli. Che autore gli si potrebbe consigliare?

```
select distinct NomeAutore from BIBLIOTECA_AUTORE
  where Titolo in
    (select Titolo from BIBLIOTECA
     where NomeCategoria in
       (select distinct NomeCategoria from BIBLIOTECA
        where Titolo in
          (select Titolo
           from BIBLIOTECA_PRESTITO bc
            where BC.Name = 'FRED FULLER')));
```

A prima vista, questa query potrebbe sembrare assai complessa; in realtà, se si analizza il codice in dettaglio diventa semplice da seguire. Si parta dalla query più interna: si ottenga un elenco dei titoli che Fred Fuller ha già letto. Per questi titoli, si faccia riferimento alla tabella BIBLIOTECA per avere un elenco dei valori di NomeCategoria distinti ai quali sono assegnati questi libri. Si ritorni alla tabella BIBLIOTECA per ricavare tutti i titoli in queste categorie. Per questi titoli, si faccia riferimento alla tabella BIBLIOTECA\_AUTORE per creare un elenco degli autori. Ecco i risultati:

NOMEAUTORE

---

-----  
BERNARD DE VOTO  
BERYL MARKHAM  
DANIEL BOORSTIN  
DAVID MCCULLOUGH  
DIETRICH BONHOEFFER  
G. B. TALBOT  
JOHN ALLEN PAULOS

MERIWETHER LEWIS  
STEPHEN AMBROSE  
STEPHEN JAY GOULD  
WILLIAM CLARK

Fred chiede dei consigli su autori nuovi, quindi si devono escludere quelli che ha già letto. Per visualizzare il nome degli autori che Fred ha già letto, è sufficiente eseguire la query sulle tabelle BIBLIOTECA\_PRESTITO e BIBLIOTECA\_AUTORE:

```
select distinct NomeAutore
  from BIBLIOTECA_AUTORE ba, BIBLIOTECA_PRESTITO bc
 where ba.Titolo = bc.Titolo
   and bc.Nome = 'FRED FULLER';
```

NOMEAUTORE

-----  
DAVID MCCULLOUGH

Ora si dovrà escludere il nome di questo autore dall'elenco che si sta per fornire. Aggiungendo un'altra clausola **and** alla query si può ottenere questo risultato:

```
select distinct NomeAutore from BIBLIOTECA_AUTORE
  where Titolo in
    (select Titolo from BIBLIOTECA
      where NomeCategoria in
        (select distinct NomeCategoria from BIBLIOTECA
          where Titolo in
            (select Titolo
              from BIBLIOTECA_PRESTITO.bc
              where BC.Nome = 'FRED FULLER'))))
  and NomeAutore not in
    (select NomeAutore
      from BIBLIOTECA_AUTORE ba, BIBLIOTECA_PRESTITO bc
      where ba.Titolo = bc.Titolo
        and bc.Nome = 'FRED FULLER');
```

NOMEAUTORE

-----  
BERNARD DE VOTO  
BERYL MARKHAM  
DANIEL BOORSTIN  
DIETRICH BONHOEFFER  
G. B. TALBOT  
JOHN ALLEN PAULOS  
MERIWETHER LEWIS  
STEPHEN AMBROSE  
STEPHEN JAY GOULD  
WILLIAM CLARK

Questa clausola **and** fa parte della query principale anche se segue la subquery. Inoltre, si noti che per alcune tabelle la query viene effettuata in più punti all'interno dello script; ciascuna di queste query viene considerata come un accesso separato della tabella.

## Uso di EXISTS e della sua subquery correlata

EXISTS effettua un controllo dell'esistenza. È posto nel modo in cui IN potrebbe essere collocato in una subquery, ma ne differisce poiché è un test logico per la restituzione di righe da una query e non per le righe in quanto tali.

Quanti autori hanno scritto più di un libro tra quelli riposti nella libreria?

```
select NomeAutore, COUNT(*)
  from BIBLIOTECA_AUTORE
 group by NomeAutore
 having COUNT(*) > 1;
```

NOMEAUTORE	COUNT(*)
DAVID MCCULLOUGH	2
DIETRICH BONHOEFFER	2
E. B. WHITE	2
SOREN KIERKEGAARD	2
STEPHEN JAY GOULD	2
W. P. KINSELLA	2
WILTON BARNHARDT	2

Tuttavia, il tentativo di trovare sia il NomeAutore sia il Titolo non riesce, perché l'istruzione group by, resa necessaria da COUNT(\*), è impostata sulla chiave primaria della tabella BIBLIOTECA\_AUTORE (NomeAutore, Titolo). Dato che per definizione ogni chiave primaria identifica in modo univoco soltanto una riga, il numero dei titoli per questa riga non può mai essere maggiore di uno, perciò il test della clausola having risulta sempre falso, e la query non trova alcuna riga:

```
no rows selected.
```

EXISTS fornisce una soluzione. La subquery seguente chiede se, per ogni NomeAutore selezionato nella query esterna, esiste un NomeAutore nella tabella BIBLIOTECA\_AUTORE con un numero di Titoli maggiore di uno. Se per un dato nome la risposta è positiva, il test di EXISTS è vero e la query esterna seleziona un NomeAutore e un Titolo. I nomi degli autori sono correlati dallo pseudonimo "BA" dato alla prima tabella BIBLIOTECA\_AUTORE.

```
select NomeAutore, Titolo
  from BIBLIOTECA_AUTORE BA
 where EXISTS
       (select *
        from BIBLIOTECA_AUTORE BA2
         where BA.NomeAutore = BA2.NomeAutore
           group by BA2.NomeAutore
          having COUNT(BA2.Titolo) > 1)
 order by NomeAutore, Titolo;
```

NOMEAUTORE	TITOLO
DAVID MCCULLOUGH	JOHN ADAMS
DAVID MCCULLOUGH	TRUMAN
DIETRICH BONHOEFFER	LETTERS AND PAPERS FROM PRISON
DIETRICH BONHOEFFER	THE COST OF DISCIPLESHIP
E. B. WHITE	CHARLOTTE'S WEB

E. B. WHITE	TRUMPET OF THE SWAN
SOREN KIERKEGAARD	EITHER/OR
SOREN KIERKEGAARD	KIERKEGAARD ANTHOLOGY
STEPHEN JAY GOULD	THE MISMEASURE OF MAN
STEPHEN JAY GOULD	WONDERFUL LIFE
W. P. KINSELLA	BOX SOCIALS
W. P. KINSELLA	SHOELESS JOE
WILTON BARNHARDT	EMMA WHO SAVED MY LIFE
WILTON BARNHARDT	GOSPEL

Le due query sono correlate: si osservi che la subquery fa riferimento alla colonna BA.NomeAutore anche se questa colonna si trova nella query esterna, e non nella subquery. All'interno della subquery, l'alias BA2 non è necessario tuttavia aiuta a semplificare il codice da mantenere.

Questa stessa query poteva essere costruita utilizzando IN e un test sulla colonna Nome. In questo caso, non è necessaria una sottoquery correlata:

```
select NomeAutore, Titolo
  from BIBLIOTECA_AUTORE BA
 where NomeAutore in
   (select NomeAutore
      from BIBLIOTECA_AUTORE
     group by NomeAutore
    having COUNT(Titolo) = 1)
 order by NomeAutore, Titolo;
```

## 12.2 Outer join

La sintassi per gli outer join è cambiata notevolmente in Oracle9i. Nei prossimi esempi, verrà proposta sia la sintassi di Oracle9i sia quella precedente a Oracle9i. Quest'ultima è ancora supportata in Oracle9i, ma il suo utilizzo non dovrebbe essere continuo. Lo sviluppo nuovo dovrebbe utilizzare la sintassi nuova. La nuova sintassi è conforme agli standard SQL ANSI, mentre quella vecchia no.

### Sintassi precedente a Oracle9i per gli outer join

Quali libri sono stati presi in prestito durante il periodo tracciato nella tabella BIBLIOTECA\_PRESTITO?

```
select distinct Titolo
  from BIBLIOTECA_PRESTITO;
-----
```

TITOLO
ANNE OF GREEN GABLES
EITHER/OR
GOOD DOG, CARL
HARRY POTTER AND THE GOBLET OF FIRE
INNUMERACY
JOHN ADAMS
MIDNIGHT MAGIC

---

MY LEDGER  
 POLAR EXPRESS  
 THE DISCOVERERS  
 THE MISMEASURE OF MAN  
 THE SHIPPING NEWS  
 TO KILL A MOCKINGBIRD  
 TRUMAN  
 WEST WITH THE NIGHT  
 WONDERFUL LIFE

Questo report è corretto, tuttavia non mostra i conteggi 0, ossia i libri che non sono stati presi in prestito. Se è necessario visualizzare l'inventario di tutti i libri insieme alla lista di controllo, si dovrà unire tramite join la tabella BIBLIOTECA\_PRESTITO alla tabella BIBLIOTECA:

```
select distinct B.Titolo
  from BIBLIOTECA_PRESTITO BC, BIBLIOTECA B
 where BC.Titolo = B.Titolo;
```

Questa query restituirà esattamente gli stessi record, ovvero le uniche righe di BIBLIOTECA che possono soddisfare i criteri di join sono quelle che sono state controllate. Per elencare gli altri libri rimanenti, si dovrà ricorrere a un *outer join*, che indichi a Oracle di restituire una riga anche se il join non produce una corrispondenza. Prima di Oracle9i, la sintassi per un outer join usava un segno più (+) sul lato del join che restituiva altre righe. In questo caso, si tratta di BIBLIOTECA\_PRESTITO. La query seguente mostra il numero massimo di giorni in cui ogni libro è stato preso in prestito:

```
select B.Titolo, MAX(BC.DataRestituzione - BC.DataPrestito)
      "MAX GIORNI PRESTITO"
  from BIBLIOTECA_PRESTITO BC, BIBLIOTECA B
 where BC.Titolo (+) = B.Titolo
 group by B.Titolo;
```

TITOLO	Max Giorni Prestito
ANNE OF GREEN GABLES	18
BOX SOCIALS	
CHARLOTTE'S WEB	
COMPLETE POEMS OF JOHN KEATS	
EITHER/OR	8
EMMA WHO SAVED MY LIFE	
GOOD DOG, CARL	14
GOSPEL	
HARRY POTTER AND THE GOBLET OF FIRE	11
INNUMERACY	21
JOHN ADAMS	28
JOURNALS OF LEWIS AND CLARK	
KIERKEGAARD ANTHOLOGY	
LETTERS AND PAPERS FROM PRISON	
MIDNIGHT MAGIC	14
MY LEDGER	16
POLAR EXPRESS	14
PREACHING TO HEAD AND HEART	
RUNAWAY BUNNY	
SHOELESS JOE	

THE COST OF DISCIPLESHIP	
THE DISCOVERERS	48
THE GOOD BOOK	
THE MISMEASURE OF MAN	31
THE SHIPPING NEWS	59
TO KILL A MOCKINGBIRD	14
TRUMAN	19
TRUMPET OF THE SWAN	
UNDER THE EYE OF THE CLOCK	
WEST WITH THE NIGHT	48
WONDERFUL LIFE	31

Tutti i Titoli contenuti in BIBLIOTECA vengono restituiti, anche quelli che non soddisfano i criteri di join. Se invece si visualizzano i valori di BIBLIOTECA\_PRESTITO.Titolo, si noterà che si tratta di valori NULL. Si provi a considerare il segno (+), che deve trovarsi subito dopo la colonna di join della tabella più breve, come un'indicazione “aggiungere un'altra riga (NULL) di BC.Titolo ogni volta che non esiste una corrispondenza con B.Titolo.”

### Sintassi di Oracle9i per gli outer join

A partire da Oracle9i, per gli outer join si potrà ricorrere alla sintassi standard ANSI SQL. Nella clausola from è possibile indicare a Oracle di eseguire un join left, right o un full outer. Si riprenda l'esempio dell'ultimo paragrafo:

```
select B.Titolo, MAX(BC.DataRestituzione - BC.DataPrestito)
      "Max Giorni Prestito"
   from BIBLIOTECA_PRESTITO BC, BIBLIOTECA B
  where BC.Titolo (+) = B.Titolo
 group by B.Titolo;
```

In questo caso, la tabella BIBLIOTECA\_PRESTITO sta ricevendo delle righe restituite durante il join anche se non sono state trovate delle corrispondenze. Questa query può essere riscritta come:

```
select B.Titolo, MAX(BC.DataRestituzione - BC.DataPrestito)
      "Max Giorni Prestito"
   from BIBLIOTECA_PRESTITO BC right outer join BIBLIOTECA B
     on BC.Titolo= B.Titolo
  group by B.Titolo;
```

Si osservi l'uso della clausola on come parte integrante della sintassi dell'outer join. Si osservi che:

```
from BIBLIOTECA_PRESTITO BC right outer join BIBLIOTECA B
```

è equivalente alla query:

```
from BIBLIOTECA B left outer join BIBLIOTECA_PRESTITO BC
```

È possibile sostituire la clausola on con una clausola using insieme al nome della colonna comune alle tabelle, non si qualifichi il nome della colonna con un nome di tabella o con un alias di tabella.

```
select Titolo, MAX(BC.DataRestituzione - BC.DataPrestito)
      "Max Giorni Prestito"
  from BIBLIOTECA_PRESTITO BC right outer join BIBLIOTECA B
    using (Titolo)
   group by Titolo;
```

Si osservi che non è consentito specificare un alias di tabella per le colonne elencate nella clausola using, così come nelle clausole group by e select.

Come accadeva con la vecchia sintassi, il lato utilizzato come tabella di controllo per l'outer join fa differenza; l'esecuzione di un'outer join a sinistra non restituirà tutti i titoli.

```
select B.Titolo, MAX(BC.DataRestituzione - BC.DataPrestito)
      "Max Giorni Prestito"
  from BIBLIOTECA_PRESTITO BC left outer join BIBLIOTECA B
    on BC.Titolo = B.Titolo
   group by B.Titolo;
```

TITOLO	Max Giorni Prestito
ANNE OF GREEN GABLES	18
EITHER/OR	8
GOOD DOG, CARL	14
HARRY POTTER AND THE GOBLET OF FIRE	11
INNUMERACY	21
JOHN ADAMS	28
MIDNIGHT MAGIC	14
MY LEDGER	16
POLAR EXPRESS	14
THE DISCOVERERS	48
THE MISMEASURE OF MAN	31
THE SHIPPING NEWS	59
TO KILL A MOCKINGBIRD	14
TRUMAN	19
WEST WITH THE NIGHT	48
WONDERFUL LIFE	31

16 rows selected.

La terza opzione, full outer join, restituisce tutte le righe da entrambe le tabelle. Le righe che non soddisfano la condizione on restituiscono valori NULL.

```
select B.Titolo, MAX(BC.DataRestituzione - BC.DataPrestito)
      "Max Giorni Prestito"
  from BIBLIOTECA_PRESTITO BC full outer join BIBLIOTECA B
    on BC.Titolo = B.Titolo
   group by B.Titolo;
```

Prima di Oracle9i, era possibile generare i risultati di un outer join completo eseguendo due outer join separati, utilizzando ogni tabella come tabella esterna, e un'operazione join per combinare i risultati in un'unica query.

## Sostituzione di NOT IN con un outer join

I vari test logici che possono essere effettuati in una clausola where hanno tutti differenti valori di prestazioni. Un test NOT IN potrebbe richiedere una lettura completa della tabella nell'istru-

zione select della subquery. Per esempio, quali libri non sono stati presi in prestito? Si potrebbe scrivere una query come la seguente:

```
select Titolo
  from BIBLIOTECA
 where Titolo not in
       (select Titolo from BIBLIOTECA_PRESTITO)
order by Titolo;
```

TITOLO
-----
BOX SOCIALS
CHARLOTTE'S WEB
COMPLETE POEMS OF JOHN KEATS
EMMA WHO SAVED MY LIFE
GOSPEL
JOURNALS OF LEWIS AND CLARK
KIERKEGAARD ANTHOLOGY
LETTERS AND PAPERS FROM PRISON
PREACHING TO HEAD AND HEART
RUNAWAY BUNNY
SHOELESS JOE
THE COST OF DISCIPLESHIP
THE GOOD BOOK
TRUMPET OF THE SWAN
UNDER THE EYE OF THE CLOCK

Questo è il modo in cui verrebbe generalmente scritta una query di questo tipo, anche se gli utenti Oracle esperti sanno che così risulterebbe lenta: infatti si potrebbe indurre Oracle a eseguire una scansione intera di tabella in poco tempo sulla tabella BIBLIOTECA\_PRESTITO. La query seguente utilizza un outer join e produce lo stesso risultato. La differenza è che quest'ultima sarà efficiente poiché l'ottimizzatore può sfruttare gli indici nelle colonne di join:

```
select distinct B.Titolo
  from BIBLIOTECA_PRESTITO BC right outer join BIBLIOTECA B
    on BC.Titolo = B.Titolo
   where BC.Titolo is NULL
order by B.Titolo;
```

TITOLO
-----
BOX SOCIALS
CHARLOTTE'S WEB
COMPLETE POEMS OF JOHN KEATS
EMMA WHO SAVED MY LIFE
GOSPEL
JOURNALS OF LEWIS AND CLARK
KIERKEGAARD ANTHOLOGY
LETTERS AND PAPERS FROM PRISON
PREACHING TO HEAD AND HEART
RUNAWAY BUNNY
SHOELESS JOE
THE COST OF DISCIPLESHIP
THE GOOD BOOK
TRUMPET OF THE SWAN
UNDER THE EYE OF THE CLOCK

Perché questa query funziona e dà gli stessi risultati che con NOT IN? L'outer join tra le due tabelle garantisce che tutte le righe siano disponibili per il test, comprese quelle dei Titoli per i quali non è elencato alcun record di controllo nella tabella BIBLIOTECA\_PRESTITO. Si consideri la riga seguente:

```
where BC.Titolo is NULL
```

essa produce solamente quei Titoli che non appaiono nella tabella BIBLIOTECA\_PRESTITO (quindi, Oracle li restituisce come titoli NULL). La logica in questo caso è oscura, ma funziona. Il modo migliore per utilizzare questa tecnica consiste nel seguire semplicemente il modello. È opportuno salvare quest'ultimo modello, inserendo una gran quantità di commenti chiarificatori accanto, per utilizzarlo quando è necessario effettuare una ricerca in una grande tabella con un NOT IN.

## Sostituzione di NOT IN con NOT EXISTS

Un modo più comune di effettuare questo tipo di query richiede l'uso della clausola NOT EXISTS. Questa clausola viene impiegata tipicamente per determinare quali valori di una tabella non hanno valori corrispondenti in un'altra. Nel suo impiego è identica alla clausola EXISTS; negli esempi seguenti si vedrà la differenza nella logica della query e i record restituiti.

NOT EXISTS consente di utilizzare una subquery correlata per eliminare da una tabella tutti i record che potrebbero essere uniti tramite join a un'altra tabella. Per questo esempio, ciò significa che è possibile eliminare dalla tabella BIBLIOTECA tutti i Titoli che sono presenti nella colonna Titolo della tabella BIBLIOTECA\_PRESTITO. La query seguente illustra come fare:

```
select B.Titolo
  from BIBLIOTECA B
 where not exists
       (select 'x' from BIBLIOTECA_PRESTITO BC
        where BC.Titolo = B.Titolo)
 order by B.Titolo;
```

TITOLI

---

BOX SOCIALS
CHARLOTTE'S WEB
COMPLETE POEMS OF JOHN KEATS
EMMA WHO SAVED MY LIFE
GOSPEL
JOURNALS OF LEWIS AND CLARK
KIERKEGAARD ANTHOLOGY
LETTERS AND PAPERS FROM PRISON
PREACHING TO HEAD AND HEART
RUNAWAY BUNNY
SHOELESS JOE
THE COST OF DISCIPLESHIP
THE GOOD BOOK
TRUMPET OF THE SWAN
UNDER THE EYE OF THE CLOCK

Questa query indica i libri che non sono stati presi in prestito, come mostrato in precedenza tramite i metodi di not in e di outer join. Ma come funziona questa query?

Per ogni record nella tabella BIBLIOTECA, viene controllata la subquery NOT EXISTS. Se il join di questo record alla tabella BIBLIOTECA\_PRESTITO restituisce una riga, la subquery ha esito positivo. NOT EXISTS comunica alla query di invertire il codice restituito; perciò, qualsiasi riga in BIBLIOTECA che può essere unita tramite a join a BIBLIOTECA\_PRESTITO non viene restituita dalla query esterna. Le uniche righe rimaste sono quelle di BIBLIOTECA che non hanno una riga corrispondente in BIBLIOTECA\_PRESTITO.

NOT EXISTS è un modo molto efficiente per effettuare questo tipo di query, soprattutto quando per il join vengono utilizzate più colonne. Dato che impiega un join, NON EXISTS è spesso in grado di sfruttare gli indici disponibili, a differenza di NOT IN. La possibilità di utilizzare indici per questo tipo di query può avere un impatto significativo sulle prestazioni della query.

### 12.3 Join naturali e interni

A partire da Oracle9*i*, con la parola chiave natural è possibile indicare che un join dovrebbe essere eseguito in base a tutte le colonne che hanno lo stesso nome nelle due tabelle che vengono unite. Per esempio, quali titoli in LIBRO\_ORDINE corrispondono a quelli già presenti in BIBLIOTECA?

```
select Titolo
  from LIBRO_ORDINE natural join BIBLIOTECA;
-----  
TITOLO  
-----  
GOSPEL  
SHOELESS JOE
```

Il join naturale ha restituito i risultati come se si fosse digitato:

```
select BO.Titolo
  from LIBRO_ORDINE BO, BIBLIOTECA
 where BO.Titolo = BIBLIOTECA.Titolo
   and BO.Editore = BIBLIOTECA.Editore
   and BO.NomeCategoria = BIBLIOTECA.NomeCategoria;
```

Il join è stato eseguito in base alle colonne comuni alle due tabelle.

Il supporto per la sintassi dei join interni è stato introdotto con Oracle9*i*. I join interni sono quelli predefiniti, restituiscano le righe comuni alle due tabelle e rappresentano l'alternativa agli outer join. Si osservi che questi join supportano le clausole on e using, quindi è possibile specificare i propri criteri di join come mostrato nel listato seguente:

```
select BO.Titolo
  from LIBRO_ORDINE BO inner join BIBLIOTECA B
    on BO.Titolo= B.Titolo;
-----  
TITOLO  
-----  
GOSPEL  
SHOELESS JOE
```

## 12.4 UNION, INTERSECT e MINUS

Spesso è necessario combinare informazioni di tipo simile da più di una tabella. Un esempio classico potrebbe essere quello di due o più elenchi di indirizzi che devono essere uniti per una campagna di spedizione. A seconda dello scopo della particolare spedizione, si potrebbero voler inviare lettere a una delle seguenti combinazioni di persone:

- A tutte quelle dei due elenchi (evitando di inviare due lettere a chi si trova in entrambi).
- Solo a quelle che sono in entrambi gli elenchi.
- A quelle di uno solo degli elenchi.

Queste tre combinazioni degli elenchi sono note in Oracle come UNION, INTERSECT e MINUS. Nei prossimi esempi si imparerà a utilizzare queste tre clausole per gestire i risultati di più query. Questi esempi metteranno a confronto i libri già disponibili (BIBLIOTECA) con quelli in ordine (LIBRO\_ORDINE).

Per visualizzare tutti i libri, occorre unire tramite join le due tabelle con la clausola UNION. Per ridurre la dimensione dell'output, vengono selezionate soltanto le voci di BIBLIOTECA dalla prima metà dell'alfabeto:

```
select Titolo from BIBLIOTECA
  where Titolo < 'M%';
```

si ottengono 14 righe:

```
select Titolo from LIBRO_ORDINE;
```

si ottengono 6 righe. Se si uniscono le due query con UNION, quante righe verranno restituite?

```
select Titolo from BIBLIOTECA
  where Titolo < 'M%'
union
select Titolo from LIBRO_ORDINE;
```

TITOLO

```
-----  
ANNE OF GREEN GABLES  
BOX SOCIALS  
CHARLOTTE'S WEB  
COMPLETE POEMS OF JOHN KEATS  
EITHER/OR  
EMMA WHO SAVED MY LIFE  
GALILEO'S DAUGHTER  
GOOD DOG, CARL  
GOSPEL  
HARRY POTTER AND THE GOBLET OF FIRE  
INNUMERACY  
JOHN ADAMS  
JOURNALS OF LEWIS AND CLARK  
KIERKEGAARD ANTHOLOGY  
LETTERS AND PAPERS FROM PRISON  
LONGITUDE  
ONCE REMOVED  
SHOELESS JOE  
SOMETHING SO STRONG
```

19 rows selected.

Dove è finito il record in più? Il problema è che uno dei valori di Titolo nella tabella LIBRO\_ORDINE si trova già nella tabella BIBLIOTECA. Per visualizzare i duplicati, occorre utilizzare UNION ALL al posto di UNION:

```
select Titolo from BIBLIOTECA
  where Titolo < 'M%'
union all
select Titolo from LIBRO_ORDINE
  order by Titolo;
```

TITOLO
-----
ANNE OF GREEN GABLES
BOX SOCIALS
CHARLOTTE'S WEB
COMPLETE POEMS OF JOHN KEATS
EITHER/OR
EMMA WHO SAVED MY LIFE
GALILEO'S DAUGHTER
GOOD DOG, CARL
GOSPEL
GOSPEL
HARRY POTTER AND THE GOBLET OF FIRE
INNUMERACY
JOHN ADAMS
JOURNALS OF LEWIS AND CLARK
KIERKEGAARD ANTHOLOGY
LETTERS AND PAPERS FROM PRISON
LONGITUDE
ONCE REMOVED
SHOELESS JOE
SOMETHING SO STRONG

20 rows selected.

Ora, il titolo doppio è elencato due volte.

Nel listato seguente, viene effettuata l'intersezione dei due elenchi di libri. Questo elenco contiene solamente i nomi contenuti in *entrambe* le tabelle di base (si osservi che in questo esempio la limitazione imposta a Titolo < 'M%' è stata eliminata):

```
select Titolo from BIBLIOTECA
  intersect
select Titolo from LIBRO_ORDINE
  order by Titolo;
```

TITOLO
-----
GOSPEL
SHOELESS JOE

Successivamente, viene creato l'elenco dei nuovi libri (in LIBRO\_ORDINE ma non in BIBLIOTECA) tramite l'operatore MINUS:

---

```
select Titolo from LIBRO_ORDINE
minus
select Titolo from BIBLIOTECA
order by Titolo;
```

TITOLO

---

```
-----  
GALILEO'S DAUGHTER  
LONGITUDE  
ONCE REMOVED  
SOMETHING SO STRONG
```

Si sarebbe potuto utilizzare MINUS anche per mostrare quali libri non erano stati presi in prestito:

```
select Titolo from BIBLIOTECA
minus
select Titolo from BIBLIOTECA_PRESTITO;
```

TITOLO

---

```
-----  
BOX SOCIALS  
CHARLOTTE'S WEB  
COMPLETE POEMS OF JOHN KEATS  
EMMA WHO SAVED MY LIFE  
GOSPEL  
JOURNALS OF LEWIS AND CLARK  
KIERKEGAARD ANTHOLOGY  
LETTERS AND PAPERS FROM PRISON  
PREACHING TO HEAD AND HEART  
RUNAWAY BUNNY  
SHOELESS JOE  
THE COST OF DISCIPLESHIP  
THE GOOD BOOK  
TRUMPET OF THE SWAN  
UNDER THE EYE OF THE CLOCK
```

15 rows selected.

Fin qui sono stati trattati gli aspetti fondamentali di UNION, INTERSECT e MINUS. Ora è opportuno un esame più dettagliato. Nella combinazione di due tabelle, Oracle non si occupa dei nomi delle colonne ai due lati dell'operatore di combinazione, ovvero richiede che ogni istruzione select sia valida e abbia colonne valide per le proprie tabelle; tuttavia i nomi delle colonne nella prima istruzione select non devono necessariamente coincidere con quelli della seconda. Oracle impone queste condizioni essenziali:

- Le istruzioni select devono avere lo stesso numero di colonne. Se in due tabelle sottoposte a query il numero di colonne selezionate è diverso, si potranno selezionare le stringhe al posto delle colonne per far coincidere gli elenchi delle colonne delle due query.
- Le colonne corrispondenti nelle istruzioni select devono appartenere allo stesso tipo di dati (non è necessario che siano della stessa lunghezza).

Quando ordina l'output, Oracle utilizza i nomi delle colonne tratti dalla prima istruzione select per assegnare i risultati della query. Di conseguenza, solo i nomi delle colonne tratti dalla prima istruzione select potranno essere impiegati in order by.

Gli operatori di combinazione possono essere utilizzati con due o più tabelle, e in questo caso la precedenza diventa un vero problema, soprattutto se compaiono `INTERSECT` e `MINUS`. Per imporre l'ordine desiderato occorre utilizzare le parentesi.

## Subquery IN

Gli operatori di combinazione possono essere utilizzati nelle subquery, ma occorre prestare molta attenzione alla precedenze. Una query del tipo:

```
select ColA from TABELLA_A
where ColA in
(select Col1 from TABELLA_1)
union
(select Col2 from TABELLA_2);
```

è scarna e ambigua. Quale operazione verrà eseguita per prima, il join delle due query come parte integrante di un'unica clausola `where`, oppure la clausola `in` che si basa sulla query di `TABELLA_1` seguita poi da un join di questo risultato con `TABELLA_2`? Per chiarire le proprie intenzioni e per far rispettare la precedenza corretta delle operazioni sono necessarie delle parentesi. Alla clausola `in` viene sempre data una precedenza maggiore rispetto al `join`, a meno che non si inseriscano delle parentesi per alterare il modo in cui viene eseguita la query. Se si desidera che il `join` abbia una precedenza maggiore, occorre utilizzare delle parentesi:

```
select ColA from TABELLA_A
where ColA in
(select Col1 from TABELLA_1
union
select Col2 from TABELLA_2);
```

## Restrizioni su UNION, INTERSECT e MINUS

Le query che utilizzano `UNION`, `INTERSECT` o `MINUS` nella clausola `where` devono avere nelle istruzioni `select` lo stesso numero e tipo di colonne. Si osservi che la costruzione `IN` equivalente non è soggetta a questa limitazione.

L'uso degli operatori di combinazione al posto di `IN`, `AND` e `OR` è una questione di stile personale. La maggior parte degli utenti SQL considera `IN`, `AND` e `OR` più chiari e più facili da capire rispetto agli operatori di combinazione.

## Capitolo 13

# Alcune possibilità complesse

- 13.1 **Raggruppamenti complessi**
- 13.2 **Uso di tabelle temporanee**
- 13.3 **Uso di ROLLUP, GROUPING e CUBE**
- 13.4 **Alberi genealogici e connect by**

Questo capitolo prosegue lo studio delle funzioni e delle caratteristiche più complesse di Oracle. Di particolare interesse è la creazione di query semplici e di gruppo che possono essere presentate in viste, l'utilizzo di totali nei calcoli e la creazione di report che mostrano strutture ad albero. Come quelle del Capitolo 12, queste tecniche non sono essenziali per la maggior parte delle situazioni in cui serve realizzare un report. Se sembrano troppo difficili, non c'è da preoccuparsi. Per chi sta iniziando ora a utilizzare Oracle e le sue funzionalità per le query, è sufficiente sapere che queste tecniche esistono e che possono essere applicate in caso di necessità.

### 13.1 Raggruppamenti complessi

È possibile costruire viste basate l'una sull'altra. Nel Capitolo 11 è stato introdotto il concetto di creazione di una vista di un raggruppamento di righe da una tabella. Come si è visto nel Capitolo 11, esiste la possibilità di unire (tramite join) facilmente le viste ad altre viste e tabelle per produrre altre viste in modo da semplificare le operazioni di realizzazione di query e report.

Via via che i raggruppamenti diventano più complessi, si scoprirà che le viste sono davvero un aiuto prezioso per le operazioni di codifica; infatti, semplificano la rappresentazione dei dati a livelli di raggruppamenti diversi all'interno dell'applicazione. Inoltre, rendono più semplice l'uso delle funzioni statiche più avanzate disponibili in Oracle9i.

Si consideri la vista CATEGORIA\_CONTEGGIO, incontrata per la prima volta nel Capitolo 11:

```
create or replace view CATEGORIA_CONTEGGIO as
select NomeCategoria, COUNT(*) as Contatore
  from BIBLIOTECA
 group by NomeCategoria;
```

```
select * from CATEGORIA_CONTEGGIO;
```

NOMECategoria	CONTATORE
NARRADULTI	6
SAGGIADULTI	10
CONSADULTI	6
NARRRAGAZZI	5
SAGGIRAGAZZI	1
ILLUSRAGAZZI	3

Si provi a ordinare i risultati in base ai valori della colonna Contatore, partendo dal più grande:

```
select * from CATEGORIA_CONTEGGIO order by Contatore desc;
```

NOME CATEGORIA	CONTATORE
SAGGIADULTI	10
NARRADULTI	6
CONSADULTI	6
NARRRAGAZZI	5
ILLUSRAGAZZI	3
SAGGIRAGAZZI	1

L'output mostra la graduatoria delle categorie; la categoria SAGGIADULTI è prima per quanto riguarda il numero dei libri. Anche senza visualizzare questo elenco, sarebbe possibile stabilire il posto occupato nelle graduatorie da un valore di Contatore differente. A questo scopo si utilizza la funzione RANK. Come mette in luce il listato seguente, la funzione RANK prende un valore come input e ha diverse clausole, tra cui *within group* e una clausola *order by* che indica a Oracle come impostare la graduatoria. Che posto occuperebbe in una graduatoria un valore di Contatore pari a 3?

```
select GRADUATORIA(3) within group
(order by Contatore desc)
from CATEGORIA_CONTEGGIO;

GRADUATORIA(3)WITHINGROUP(ORDERBYCONTATOREDESC)
-----
```

5

Un valore pari a 3 per Contatore sarebbe il quinto valore più alto di Contatore. E un valore di Contatore pari a 8?

```
select GRADUATORIA(8) within group
(order by Contatore desc)
from CATEGORIA_CONTEGGIO;

GRADUATORIA(8)WITHINGROUP(ORDERBYCONTATOREDESC)
-----
```

2

L'aggiunta di questi cinque libri alla categoria la sposterebbe al secondo posto. Su una base percentile, quale sarebbe la graduatoria percentile per questa categoria?

```
select PERCENTO_GRADUATORIA(8) within group
(order by Contatore desc)
from CATEGORIA_CONTEGGIO;

PERCENTO_GRADUATORIA(8)WITHINGROUP(ORDERBYCONTATOREDESC)
-----
```

.166666667

Come previsto, sarebbe compresa nei primi sei delle graduatorie.

Con questa tecnica di associare viste di summary e funzioni analitiche è possibile creare viste e report che includono medie ponderate, rendimenti effettivi, percentuali di totali e di totali parziali e molti altri calcoli simili. Non esiste alcun limite alle viste che possono essere costruite in base ad altre, anche se i calcoli più complessi raramente richiedono più di tre o quattro livelli di viste costruite su altre viste. Si osservi inoltre che nella clausola `from` è possibile creare delle viste in linea, come illustrato nel Capitolo 11.

## 13.2 Uso di tabelle temporanee

È possibile creare una tabella che esiste soltanto per la propria sessione, oppure i cui dati persistono solo per la durata di una transazione. Le tabelle temporanee possono essere utilizzate per supportare rollup specializzati oppure requisiti specifici per l'elaborazione delle applicazioni.

Per creare una tabella temporanea, si ricorre al comando `create global temporary table`. Quando si crea una tabella temporanea, è possibile specificare se la sua durata comprende l'intera sessione (tramite la clausola `on commit preserve rows`), oppure se le sue righe dovranno essere eliminate al termine della transazione (con la clausola `on commit delete rows`).

A differenza di una tabella permanente, una tabella temporanea non alloca automaticamente dello spazio quando viene creata. Lo spazio verrà allocato per la tabella in modo dinamico mentre vengono inserite le righe:

```
create global temporary table ANNO_ROLLUP (
    Anno    NUMBER(4),
    Mese   VARCHAR2(9),
    Contatore NUMBER)
on commit preserve rows;
```

È possibile visualizzare la durata dei dati in `ANNO_ROLLUP` eseguendo la query sulla colonna Durata della tabella `UTENTE_TABELLE`. In questo caso il valore della colonna Durata è `SYSSSESSION`.

Ora che la tabella `ANNO_ROLLUP` esiste, la si può popolare, per esempio tramite un comando `insert as select` con una query complessa. Successivamente, si potrà sottoporre a query questa tabella in quanto parte di un join con altre tabelle. Questo metodo potrebbe risultare più semplice da implementare rispetto a quelli mostrati nei paragrafi precedenti.

## 13.3 Uso di ROLLUP, GROUPING e CUBE

In che modo è possibile eseguire operazioni di raggruppamento, come i totali, all'interno di un'unica istruzione SQL senza ricorrere ai comandi `SQL*Plus?` A partire da Oracle8i sono disponibili due funzioni, `ROLLUP` e `CUBE`, che servono proprio per migliorare le operazioni di raggruppamento eseguite nelle query. Si osservi come queste funzioni consentono di gestire i dati correlati alle restituzioni dei libri. Il programma di prestito dei libri gode di una popolarità sempre maggiore, quindi il periodo di prestito si è ridotto a 14 giorni e per ogni giorno in più si devono pagare 0,20 euro. Il report seguente mostra gli addebiti, dovuti a ritardi nella consegna, per persona:

```
set headsep !
column Nome format a20
```

```

column Titolo format a20 word_wrapped
column GiorniPrestito format 999.99 heading 'Giorni!Prestito'
column GiorniRitardo format 999.99 heading 'Giorni!Ritardo'

break on Nome skip 1 on report
compute sum of TassaRitardo on Nome
set linesize 80
set pagesize 60
set newpage 0

select Nome, Titolo, DataRestituzione,
       DataRestituzione-DataPrestito as GiorniPrestito /*Conta i giorni*/,
       DataRestituzione-DataPrestito -14 GiorniRitardo,
       DataRestituzione-DataPrestito -14)*0.20 TassaRitardo
  from BIBLIOTECA_PRESTITO
 where DataRestituzione-DataPrestito 14
 order by Nome, DataPrestito;

```

NOME	TITOLO	DATAREST	Giorni Prestito	Giorni Ritardo	TASSARITARDO
DORAH TALBOT	MY LEDGER	03-MAR-02	16.00	2.00	.4
*****	*****				-----
sum					.4
EMILY TALBOT	ANNE OF GREEN GABLES	20-JAN-02	18.00	4.00	.8
*****	*****				-----
sum					.8
FRED FULLER	JOHN ADAMS	01-MAR-02	28.00	14.00	2.8
	TRUMAN	20-MAR-02	19.00	5.00	1
*****	*****				-----
sum					3.8
GERHARDT KENTGEN	WONDERFUL LIFE	02-FEB-02	31.00	17.00	3.4
	THE MISMEASURE OF	05-MAR-02	20.00	6.00	1.2
	MAN				-----
*****	*****				-----
sum					4.6
JED HOPKINS	INNUMERACY	22-JAN-02	21.00	7.00	1.4
*****	*****				-----
sum					1.4
PAT LAVAY	THE MISMEASURE OF	12-FEB-02	31.00	17.00	3.4
	MAN				-----
*****	*****				-----
sum					3.4
ROLAND BRANDT	THE SHIPPING NEWS	12-MAR-02	59.00	45.00	9
	THE DISCOVERERS	01-MAR-02	48.00	34.00	6.8
	WEST WITH THE NIGHT	01-MAR-02	48.00	34.00	6.8
*****	*****				-----
sum					22.6

È possibile eliminare la visualizzazione di GiorniPrestito e concentrarsi sulle tasse di ritardo, mostrando gli importi calcolati in base alle date di restituzione:

```

clear compute
clear break
select DataRestituzione, Nome,
       SUM((DataRestituzione-DataPrestito -14)*0.20) TassaRitardo
  from BIBLIOTECA_PRESTITO
 where DataRestituzione-DataPrestito > 14
 group by DataRestituzione, Nome
 order by DataRestituzione, Nome;

```

DATAREST NOME	TASSARITARDO
20-JAN-02 EMILY TALBOT	.8
22-JAN-02 JED HOPKINS	1.4
02-FEB-02 GERHARDT KENTGEN	3.4
12-FEB-02 PAT LAVAY	3.4
01-MAR-02 FRED FULLER	2.8
01-MAR-02 ROLAND BRANDT	13.6
03-MAR-02 DORAH TALBOT	.4
05-MAR-02 GERHARDT KENTGEN	1.2
12-MAR-02 ROLAND BRANDT	9
20-MAR-02 FRED FULLER	1

e poi modificare questa tabella ulteriormente in modo da raggruppare questi importi per mese:

```

select TO_CHAR(DataRestituzione,'MESE'), Nome,
       SUM((DataRestituzione-DataPrestito -14)*0.20) TassaRitardo
  from BIBLIOTECA_PRESTITO
 where DataRestituzione-DataPrestito > 14
 group by TO_CHAR(DataRestituzione,'MESE'), Nome;

```

TO_CHAR(R NOME	TASSARITARDO
FEBRUARY GERHARDT KENTGEN	3.4
FEBRUARY PAT LAVAY	3.4
JANUARY EMILY TALBOT	.8
JANUARY JED HOPKINS	1.4
MARCH DORAH TALBOT	.4
MARCH FRED FULLER	3.8
MARCH GERHARDT KENTGEN	1.2
MARCH ROLAND BRANDT	22.6

Invece di raggruppare per Mese e per Nome, si potrebbe usare la funzione ROLLUP per generare i totali parziali e i totali. Nell'esempio seguente, la clausola group by viene modificata per includere una chiamata alla funzione ROLLUP. Si osservino le righe in più create alla fine del risultato e dopo ogni mese.

```

select TO_CHAR(DataRestituzione,'MESE'), Nome,
       SUM((DataRestituzione-DataPrestito -14)*0.20) TassaRitardo
  from BIBLIOTECA_PRESTITO
 where DataRestituzione-DataPrestito > 14
 group by ROLLUP(TO_CHAR DataRestituzione,'MESE'), Nome;

```

TO_CHAR(R NOME	TASSARITARDO
FEBRUARY GERHARDT KENTGEN	3.4
FEBRUARY PAT LAVAY	3.4

---

FEBRUARY		6.8
JANUARY	EMILY TALBOT	.8
JANUARY	JED HOPKINS	1.4
JANUARY		2.2
MARCH	DORAH TALBOT	.4
MARCH	FRED FULLER	3.8
MARCH	GERHARDT KENTGEN	1.2
MARCH	ROLAND BRANDT	22.6
MARCH		28
		37

Per ogni mese, Oracle ha calcolato la TassaRitardo totale indicandola con un valore NULL per il nome. L'output mostra nel mese di febbraio due importi separati pari a €3,40, e un totale mensile di €6,80. Per il trimestre, il totale degli importi dovuti a ritardi è di €37,00. Anche con i comandi SQLPLUS (fare riferimento al Capitolo 6) sarebbe stato possibile calcolare questi risultati, tuttavia questo metodo consente di ottenere queste somme con un solo comando SQL a prescindere dallo strumento utilizzato per eseguire la query del database.

Si provi ad abbellire il report. Con la funzione GROUPING è possibile stabilire se una riga è un totale o un totale parziale (creato da ROLLUP) oppure se corrisponde a un valore NULL nel database. Nella clausola select, la colonna Nome verrà selezionata in questo modo:

```
select DECODE(GROUPING(Nome),1, 'Tutti i nomi',Nome),
```

La funzione GROUPING restituisce un valore pari a 1 se il valore della colonna è stato generato da un'azione compiuta da ROLLUP. Questa query si serve di DECODE (analizzata in dettaglio nel Capitolo 17) per valutare il risultato della funzione GROUPING. Se l'output prodotto da GROUPING è 1, significa che il valore è stato creato dalla funzione ROLLUP, quindi Oracle visualizza la frase "Tutti i nomi"; altrimenti, visualizzerà il valore della colonna Nome. Si provi ad applicare una logica simile alla colonna Data. La query completa è illustrata nel listato seguente, insieme all'output:

```
select DECODE(GROUPING(TO_CHAR(DataRestituzione,'MESE')),1,
    'Tutti i mesi',TO_CHAR(DataRestituzione,'MESE')),
    DECODE(GROUPING(Nome),1, 'Tutti i nomi',Nome),
    SUM((DataRestituzione-DataPrestito -14)*0.20) TassaRitardo
  from BIBLIOTECA_PRESTITO
 where DataRestituzione-DataPrestito > 14
 group by ROLLUP(TO_CHAR(DataRestituzione, 'MESE'), Nome);

DECODE(GRO DECODE(GROUPING(NOME),1,' TASSARITARDO
----- -----
FEBRUARY GERHARDT KENTGEN      3.4
FEBRUARY PAT LAVAY            3.4
FEBRUARY Tutti i nomi        6.8
JANUARY  EMILY TALBOT          .8
JANUARY  JED HOPKINS          1.4
JANUARY  Tutti i nomi        2.2
MARCH   DORAH TALBOT          .4
MARCH   FRED FULLER          3.8
MARCH   GERHARDT KENTGEN      1.2
MARCH   ROLAND BRANDT         22.6
MARCH   Tutti i nomi          28
Tutti i mesi Tutti i nomi      37
```

Con la funzione CUBE è possibile ottenere i totali parziali per tutte le combinazioni dei valori nella clausola group by. La query seguente utilizza la funzione CUBE per generare queste informazioni:

```
select DECODE(GROUPING(TO_CHAR(DataRestituzione,'MESE')),1,
              'Tutti i mesi',TO_CHAR(DataRestituzione,'MESE')),
       DECODE(GROUPING(Nome),1, 'Tutti i nomi',Nome),
       SUM((DataRestituzione-DataPrestito -14)*0.20) TassaRitardo
  from BIBLIOTECA_PRESTITO
 where DataRestituzione-DataPrestito > 14
 group by CUBE(TO_CHAR(DataRestituzione, 'MESE'), Nome);

DECODE(GRO DECODE(GROUPING(NOME),1,' TASSARITARDO
-----
FEBRUARY GERHARDT KENTGEN          3.4
FEBRUARY PAT LAVAY                 3.4
FEBRUARY Tutti i nomi               6.8
JANUARY  EMILY TALBOT                .8
JANUARY  JED HOPKINS                1.4
JANUARY  Tutti i nomi               2.2
MARCH   DORAH TALBOT                .4
MARCH   FRED FULLER                 3.8
MARCH   GERHARDT KENTGEN            1.2
MARCH   ROLAND BRANDT               22.6
MARCH   Tutti nomi                  28
Tutti i mesi DORAH TALBOT           .4
Tutti i mesi EMILY TALBOT            .8
Tutti i mesi FRED FULLER             3.8
Tutti i mesi GERHARDT KENTGEN        4.6
Tutti i mesi JED HOPKINS              1.4
Tutti i mesi PAT LAVAY                3.4
Tutti i mesi ROLAND BRANDT            22.6
Tutti i mesi Tutti i nomi              37
```

La funzione CUBE ha fornito i riepiloghi generati dall'opzione ROLLUP, inoltre mostra le somme ordinate per Nome per la categoria "Tutti i mesi". La possibilità di eseguire questi riepiloghi in SQL standard migliora notevolmente la capacità di selezionare per i propri utenti il migliore strumento per la creazione di report.

## 13.4 Alberi genealogici e connect by

Una delle funzionalità più interessanti, ma poco utilizzate e capite, di Oracle è la clausola connect by. Si tratta semplicemente di un metodo per riportare in ordine i rami di un *albero genealogico*. Questi alberi si trovano in varie situazioni: nella genealogia di famiglie umane, del bestiame, dei cavalli, delle amministrazioni di società, dei reparti di aziende, di attività industriali, della letteratura, delle idee, dell'evoluzione, della ricerca scientifica, della teoria e persino di viste costruite su altre viste.

La clausola connect by fornisce uno strumento per riportare tutti i membri della famiglia in questi alberi. Consente di escludere rami o singoli membri di un albero genealogico e di muoversi attraverso l'albero verso l'alto o verso il basso, riportando i membri della famiglia incontrati lungo il percorso.

Il primo antenato nell'albero tecnicamente viene chiamato *nodo principale*. Nel linguaggio corrente corrisponderebbe al tronco. Dal tronco si estendono i rami, che hanno altri rami, che hanno ancora altri rami. Le forcille in corrispondenza delle quali uno o più rami si distaccano da un ramo più largo prendono il nome di *nodi* e l'estremità di un ramo viene chiamata *foglia* o *nodo foglia*. La Figura 13.1 mostra una rappresentazione di un albero.

Di seguito è riportata una tabella di mucche e tori nati tra il gennaio del 1900 e l'ottobre del 1908. Ciascun discendente, appena nato, viene inserito come una riga nella tabella, insieme con il sesso, i genitori (la mucca e il toro) e la data di nascita. Se si confrontano Mucca e Figlio in questa tabella con la Figura 13.1, si scopre che corrispondono. Per esempio, EVE non ha alcuna mucca o toro genitore, perché è nata in un'altra fattoria, mentre ADAM e BANDIT sono tori introdotti per la riproduzione, senza genitori nella tabella.

```
Mucca format a6
column Toro format a6
column Figlio format a10
column Sesso format a3
```

```
select * from ALLEVAMENTO
order by Datanascita;
```

FIGLIO	SESSO	MUCCA	TORO	DATANASCITA
BETSY	F	EVE	ADAM	02-JAN-00
POCO	M	EVE	ADAM	15-JUL-00
GRETA	F	EVE	BANDIT	12-MAR-01

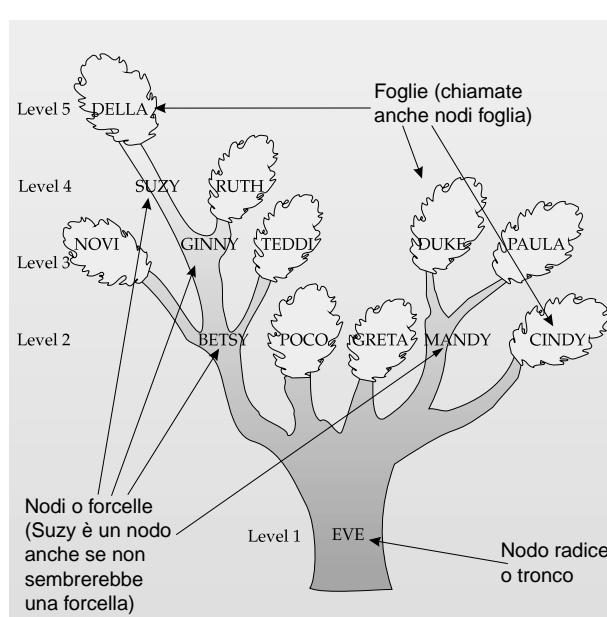


Figura 13.1 I discendenti di Eve.

MANDY	F	EVE	POCO	22-AUG-02
CINDY	F	EVE	POCO	09-FEB-03
NOVI	F	BETSY	ADAM	30-MAR-03
GINNY	F	BETSY	BANDIT	04-DEC-03
DUKE	M	MANDY	BANDIT	24-JUL-04
TEDDI	F	BETSY	BANDIT	12-AUG-05
SUZY	F	GINNY	DUKE	03-APR-06
PAULA	F	MANDY	POCO	21-DEC-06
RUTH	F	GINNY	DUKE	25-DEC-06
DELLA	F	SUZY	BANDIT	11-OCT-08
EVE	F			
ADAM	M			
BANDIT	M			

Poi viene scritta una query per illustrare visivamente le relazioni familiari. Per fare questo si utilizza LPAD e una colonna speciale, Level, che si ottiene con connect by. Level è un numero, da 1 per EVE a 5 per DELLA, che in realtà indica la *generazione*. Se EVE è la prima generazione di bovini, allora DELLA è la quinta generazione. Ogni volta che si usa la clausola connect by, la colonna Level può essere utilizzata nell'istruzione select per mostrare la generazione di ogni riga. Level è una *pseudocolonna*, come SysDate e User. Non fa realmente parte della tabella, ma è disponibile in determinate circostanze. Il prossimo listato presenta un esempio che utilizza Level.

I risultati di questa query sono visibili nella tabella seguente, ma perché l'istruzione select ha prodotto questo? Come funziona?

```
column Figlio format a30
```

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sesso, Datanascita
  from ALLEVAMENTO
 start with Figlio = 'EVE'
connect by Mucca = PRIOR Figlio;
```

MUCCA	TORO	FIGLIO	SESSO	DATANASCITA
-----				
EVE				
EVE	ADAM	BETSY	F	02-JAN-00
BETSY	ADAM	NOVI	F	30-MAR-03
BETSY	BANDIT	GINNY	F	04-DEC-03
GINNY	DUKE	SUZY	F	03-APR-06
SUZY	BANDIT		DELLA	11-OCT-08
GINNY	DUKE		RUTH	25-DEC-06
BETSY	BANDIT	TEDDI	F	12-AUG-05
EVE	ADAM	POCO	M	15-JUL-00
EVE	BANDIT	GRETA	F	12-MAR-01
EVE	POCO	MANDY	F	22-AUG-02
MANDY	BANDIT	DUKE	M	24-JUL-04
MANDY	POCO	PAULA	F	21-DEC-06
EVE	POCO	CINDY	F	09-FEB-03

Si noti che questa è in pratica la Figura 13.1 percorsa in senso orario. EVE non è al centro, ma è il nodo principale (tronco) di questo albero. I suoi figli sono BETSY, POCO, GRETA, MANDY e CINDY. I figli di BETSY sono NOVI, GINNY e TEDDI. I figli di GINNY sono SUZY e RUTH. E il figlio di SUZY è DELLA. Anche MANDY ha due figli, che si chiamano DUKE e PAULA.

Questo albero inizia con EVE come primo “discendente”. Se l’istruzione SQL avesse contenuto start with MANDY, sarebbero stati selezionati solo MANDY, DUKE e PAULA. L’istruzione start with definisce l’inizio della parte di albero da visualizzare e include solo i rami che si allungano dall’individuo da essa specificato. start with significa proprio “inizia con”.

LPAD nell’istruzione select ha probabilmente una costruzione un po’ confusa. Il formato di LPAD, riportato dal Capitolo 7, è il seguente:

```
LPAD(stringa ,lunghezza [, 'set'])
```

In altre parole, si indica di prendere la *stringa* specificata e di riempirla a sinistra con il *set* di caratteri specificato fino a raggiungere la *lunghezza* indicata. Se non viene specificato alcun set di caratteri, si riempia la stringa a sinistra con degli spazi vuoti. Occorre confrontare questa sintassi con LPAD nell’istruzione select mostrata in precedenza:

```
LPAD(' ',6*(Level-1))
```

In questo caso, la *stringa* è costituita da un singolo carattere, uno spazio (indicato dallo spazio racchiuso tra apici). 6\*(Level-1) è la *lunghezza*, e poiché set non è definito, vengono utilizzati degli spazi. In altre parole, viene comunicato a SQL di prendere questa stringa di uno spazio e di riempirla a sinistra con il numero di spazi determinato da 6\*(Level-1), un calcolo eseguito dapprima sottraendo 1 da Level e moltiplicando questo risultato per 6. Per EVE, Level è 1, quindi 6\*(1-1), oppure 0 spazi. Per BETSY, Level (la sua generazione) è 2, quindi un LPAD pari a 6. Pertanto, per ogni generazione dopo la prima, vengono concatenati sei spazi in più alla sinistra della colonna Figlio. L’effetto è evidente nel risultato appena mostrato. Il nome di ogni Figlio viene rientrato riempiendo a sinistra con un numero di spazi equivalente al suo Level o alla sua generazione.

Perché si effettuano un riempimento e una concatenazione, invece di utilizzare LPAD direttamente su Figlio? Per due motivi. Primo, un’istruzione LPAD diretta su Figlio avrebbe giustificato i nomi di Figlio a destra. I nomi di ogni livello avrebbero avuto le ultime lettere allineate verticalmente. Secondo, se Level-1 è uguale a 0, come per EVE, il risultato di LPAD per EVE sarebbe stato di 0 caratteri. EVE sarebbe così sparita:

```
select Mucca, Toro, LPAD(Figlio,6*(Level-1),' ') Figlio,
       Sesso, Datanascita from ALLEVAMENTO
  start with Figlio = 'EVE'
 connect by Mucca = PRIOR Figlio;
```

MUCCA	TORO	FIGLIO	SESSO	DATANASCITA
EVE	ADAM	BETSY	F	
BETSY	ADAM	NOVI	F	02-JAN-00
BETSY	BANDIT	GINNY	F	30-MAR-03
GINNY	DUKE	SUZY	F	04-DEC-03
SUZY	BANDIT	DELLA	F	03-APR-06
GINNY	DUKE	RUTH	F	11-OCT-08
BETSY	BANDIT	TEDDI	F	25-DEC-06
EVE	ADAM	POCO	M	12-AUG-05
EVE	BANDIT	GRETA	F	15-JUL-00
EVE	POCO	MANDY	F	12-MAR-01
MANDY	BANDIT	DUKE	F	22-AUG-02
MANDY	POCO	PAULA	M	24-JUL-04
EVE	POCO	CINDY	F	21-DEC-06
				09-FEB-03

Quindi, per ottenere la giusta spaziatura per ogni livello, per assicurarsi che EVE sia presente e per avere i nomi allineati verticalmente a sinistra, si dovrebbe utilizzare LPAD con la funzione di concatenazione e non direttamente sulla colonna Figlio.

Ma come funziona connect by? Si esamini di nuovo la Figura 13.1. A partire da NOVI e risalendo l'elenco all'indietro, quali mucche sono le discendenti che vengono prima di NOVI? La prima è BETSY e quella appena prima di BETSY è EVE. Anche se non è immediatamente comprensibile, la clausola:

```
connect by Mucca = PRIOR Figlio
```

comunica a SQL di trovare la riga successiva in cui il valore della colonna Mucca è uguale al valore della colonna Figlio nella riga precedente. Si esamini la tabella e si vedrà che è vero.

## Esclusione di individui e rami

Esistono due metodi per escludere delle mucche da un rapporto. Uno utilizza la normale tecnica della clausola where e l'altro usa proprio la clausola connect by. La differenza è che con la clausola connect by si esclude non solo la mucca citata, ma anche tutti i suoi figli. Se si adopera connect by per escludere BETSY, spariscono anche NOVI, GINNY, TEDDI, SUZY, RUTH e DELLA. connect by in realtà segue la struttura dell'albero. Se BETSY non fosse mai nata, non vi sarebbero stati nemmeno i suoi figli. In questo esempio, la clausola and modifica la clausola connect by:

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sesso, DataNascita
  from ALLEVAMENTO
 start with Figlio = 'EVE'
connect by Mucca = PRIOR Figlio
      and Figlio != 'BETSY';
```

MUCCA	TORO	FIGLIO	SESSO	DATANASCITA
	EVE		F	
EVE	ADAM	POCO	M	15-JUL-00
EVE	BANDIT	GRETA	F	12-MAR-01
EVE	POCO	MANDY	F	22-AUG-02
MANDY	BANDIT	DUKE	M	24-JUL-04
MANDY	POCO	PAULA	F	21-DEC-06
EVE	POCO	CINDY	F	09-FEB-03

La clausola where rimuove soltanto la mucca o le mucche in essa citate. Se BETSY muore, viene rimossa dal diagramma, ma i suoi figli rimangono. In effetti, si noti che BETSY è ancora sotto la colonna Mucca come madre dei suoi figli, NOVI, GINNY e TEDDI:

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sesso, DataNascita
  from ALLEVAMENTO
 where Figlio != 'BETSY'
 start with Figlio = 'EVE'
connect by Mucca = PRIOR Figlio;
```

MUCCA	TORO	FIGLIO	SESSO	DATANASCITA
		EVE	F	
BETSY	ADAM	NOVI	F	30-MAR-03
BETSY	BANDIT	GINNY	F	04-DEC-03
GINNY	DUKE	SUZY	F	03-APR-06
SUZY	BANDIT	DELLA	F	11-OCT-08
GINNY	DUKE	RUTH	F	25-DEC-06
BETSY	BANDIT	TEDDI	F	12-AUG-05
EVE	ADAM	POCO	M	15-JUL-00
EVE	BANDIT	GRETA	F	12-MAR-01
EVE	POCO	MANDY	F	22-AUG-02
MANDY	BANDIT	DUKE	M	24-JUL-04
MANDY	POCO	PAULA	F	21-DEC-06
EVE	POCO	CINDY	F	09-FEB-03

L'ordine in cui l'albero genealogico viene visualizzato quando si usa connect by è livello per livello, da sinistra a destra, come mostrato nella Figura 13.1, iniziando con il livello più basso, Level 1. Nell'esempio seguente si vuole alterare questo ordine in modo da raggruppare le mucche e i loro figli in base alla data di nascita:

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sesso, Datanascita from ALLEVAMENTO
  start with Figlio = 'EVE'
connect by Mucca = PRIOR Figlio
  order by Mucca, Datanascita;
```

MUCCA	TORO	FIGLIO	SESSO	DATANASCITA
BETSY	ADAM	NOVI	F	30-MAR-03
BETSY	BANDIT	GINNY	F	04-DEC-03
BETSY	BANDIT	TEDDI	F	12-AUG-05
EVE	ADAM	BETSY	F	02-JAN-00
EVE	ADAM	POCO	M	15-JUL-00
EVE	BANDIT	GRETA	F	12-MAR-01
EVE	POCO	MANDY	F	22-AUG-02
EVE	POCO	CINDY	F	09-FEB-03
GINNY	DUKE	SUZY	F	03-APR-06
GINNY	DUKE	RUTH	F	25-DEC-06
MANDY	BANDIT	DUKE	M	24-JUL-04
MANDY	POCO	PAULA	F	21-DEC-06
SUZY	BANDIT	DELLA	F	11-OCT-08
		EVE	F	

Le generazioni sono ancora evidenti nella visualizzazione, ma i discendenti sono raggruppati più vicini l'uno all'altro. Lo stesso albero genealogico si può esaminare anche per Datanascita, come segue:

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sesso, Datanascita
  from ALLEVAMENTO
  start with Figlio = 'EVE'
connect by Mucca = PRIOR Figlio
  order by Datanascita;
```

MUCCA	TORO	FIGLIO	SESSO	DATANASCITA
EVE	ADAM	BETSY	F	02-JAN-00
EVE	ADAM	POCO	M	15-JUL-00
EVE	BANDIT	GRETA	F	12-MAR-01
EVE	POCO	MANDY	F	22-AUG-02
EVE	POCO	CINDY	F	09-FEB-03
BETSY	ADAM	NOVI	F	30-MAR-03
BETSY	BANDIT	GINNY	F	04-DEC-03
MANDY	BANDIT	DUKE	M	24-JUL-04
BETSY	BANDIT	TEDDI	F	12-AUG-05
GINNY	DUKE	SUZY	F	03-APR-06
MANDY	POCO	PAULA	F	21-DEC-06
GINNY	DUKE	RUTH	F	25-DEC-06
SUZY	BANDIT	DELLA	F	11-OCT-08
		EVE	F	

Ora, l'ordine delle righe non mostra più le generazioni, come in un albero, ma l'indentazione conserva ancora queste informazioni. Però non è possibile stabilire la relazione tra genitori e discendenti senza osservare le colonne Mucca e Toro. Se non si conosce la Datanascita di EVE, la visualizzazione dell'albero genealogico chiaramente viene alterata.

## Spostarsi verso le radici

Finora, la direzione del movimento nella stesura del report nell'albero genealogico è stata dai genitori verso i figli. È possibile iniziare con un figlio e spostarsi verso il genitore, il genitore del genitore e così via? Per fare questo, occorre semplicemente spostare la parola prior dall'altra parte del segno di uguale. Il seguente listato riporta gli antenati di DELLA:

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
      Sesso, Datanascita
     from ALLEVAMENTO
    start with Figlio = 'DELLA'
   connect by Figlio = PRIOR Mucca;
```

MUCCA	TORO	FIGLIO	SESSO	DATANASCITA
SUZY	BANDIT	DELLA	F	11-OCT-08
GINNY	DUKE	SUZY	F	03-APR-06
BETSY	BANDIT	GINNY	F	04-DEC-03
EVE	ADAM	BETSY	F	02-JAN-00
		EVE	F	

Questo risultato mostra le radici di DELLA, ma è un po' fuorviante rispetto alle visualizzazioni precedenti. È come se DELLA fosse l'antenata ed EVE l'ultima discendente. Aggiungere un order by per Datanascita può essere d'aiuto, ma EVE compare ancora all'estrema destra:

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
      Sesso, Datanascita
     from ALLEVAMENTO
    start with Figlio = 'DELLA'
   connect by Figlio = PRIOR Mucca
   order by Datanascita;
```

MUCCA	TORO	FIGLIO	SESSO	DATANASCITA
EVE	ADAM	BETSY	F	02-JAN-00
BETSY	BANDIT	GINNY	F	04-DEC-03
GINNY	DUKE	SUZY	F	03-APR-06
SUZY	BANDIT DELLA		F	11-OCT-08
		EVE	F	

La soluzione consiste semplicemente nel cambiare il calcolo in LPAD:

```
select Mucca, Toro, LPAD(' ',6*(5-Level))||Figlio Figlio,
       Sesso, Datanascita
  from ALLEVAMENTO
 start with Figlio = 'DELLA'
connect by Figlio = PRIOR Mucca
order by Datanascita;
```

MUCCA	TORO	FIGLIO	SESSO	DATANASCITA
EVE	ADAM	BETSY	F	02-JAN-00
BETSY	BANDIT	GINNY	F	04-DEC-03
GINNY	DUKE	SUZY	F	03-APR-06
SUZY	BANDIT	DELLA	F	11-OCT-08
		EVE	F	

Infine, si osservi come questo report appare diverso quando connect by segue la discendenza di Toro. Ecco i discendenti di ADAM:

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sesso, Datanascita
  from ALLEVAMENTO
 start with Figlio = 'ADAM'
connect by PRIOR Figlio = Toro;
```

MUCCA	TORO	FIGLIO	SESSO	DATANASCITA
	ADAM		M	
EVE	ADAM	BETSY	F	02-JAN-00
EVE	ADAM	POCO	M	15-JUL-00
EVE	POCO	MANDY	F	22-AUG-02
EVE	POCO	CINDY	F	09-FEB-03
MANDY	POCO	PAULA	F	21-DEC-06
BETSY	ADAM	NOVI	F	30-MAR-03

ADAM e BANDIT erano i tori originari che hanno dato inizio alla mandria. Per creare un singolo albero che riporti i discendenti sia di ADAM sia di BANDIT, si dovrebbe inventare un “padre” per questi due tori, che sia la radice dell’albero. Uno dei vantaggi che hanno questi alberi alternativi rispetto al tipo mostrato in precedenza è che molti gruppi ereditari, dalle famiglie ai progetti, ai reparti delle aziende, possono essere rappresentati con precisione in più di un modo:

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sesso, DataNascita
  from ALLEVAMENTO
 start with Figlio = 'BANDIT'
connect by PRIOR Figlio = Toro;
```

MUCCA	TORO	FIGLIO	SESSO	DATANASCITA
BANDIT				
EVE	BANDIT	GRETA	M	
BETSY	BANDIT	GINNY	F	12-MAR-01
MANDY	BANDIT	DUKE	F	04-DEC-03
GINNY	DUKE	SUZY	M	24-JUL-04
GINNY	DUKE	RUTH	F	03-APR-06
BETSY	BANDIT	TEDDI	F	25-DEC-06
SUZY	BANDIT	DELLA	F	12-AUG-05
			F	11-OCT-08

## Le regole fondamentali

Utilizzare connect by e start with per creare report simili ad alberi non è difficile, ma occorre seguire alcune regole fondamentali:

- L'ordine delle clausole quando si usa connect by è il seguente:

```
select
  from
  where
  start
  with
  connect by
  order by
```

- prior fa in modo che il report vada dalla radice verso le foglie (se la colonna prior è il genitore) o dalle foglie verso la radice (se la colonna prior è il discendente).
- Una clausola where elimina gli individui dall'albero, ma non i loro discendenti (o gli antenati, se prior è a destra del segno di uguale).
- Una qualificazione in connect by (in particolare un “non uguale”) elimina un individuo e tutti i suoi discendenti (o antenati, a seconda di come si traccia l'albero).
- connect by non può essere utilizzata con un join di tabella nella clausola where.

Probabilmente sono in pochi a ricordare in modo esatto questa particolare serie di comandi. Tuttavia, con una conoscenza base dell'albero e dell'ereditarietà, per costruire un'istruzione select appropriata per un report su un albero è sufficiente consultare la sintassi corretta in questo capitolo.



## Capitolo 14

# Creazione di un report in SQL\*PLUS

- 14.1 **Formattazione avanzata**
- 14.2 **set termout off e set termout on**
- 14.3 **Variabili in SQLPLUS**
- 14.4 **Formattazione di numeri**
- 14.5 **Uso di mask.sql**
- 14.6 **show all e spool**
- 14.7 **Aggiunta di linee vuote**
- 14.8 **Ulteriori controlli per la realizzazione  
di report**

Nel Capitolo 6 sono stati descritti i concetti fondamentali per la formattazione di report, mentre nei Capitoli 3 e 11 sono stati forniti metodi per l'utilizzo di gruppi e viste. In questo capitolo vengono esaminati metodi di formattazione più avanzati e calcoli più complessi di medie ponderate. In questo capitolo viene presentata una rassegna di un certo numero di interdipendenze fra i vari comandi SQLPLUS.

## 14.1 Formattazione avanzata

Nel Capitolo 1 è stato mostrato un esempio di tabella di quote azionarie tratta da un giornale. Ora, questa tabella viene manipolata per dedurne informazioni non evidenti. Per brevità, le azioni esaminate sono limitate a tre industrie: elettronica, spaziale e medica. Questi sono i comandi per le colonne:

```
column Nett format 99.90
column Industria format a11
column Societa format a18
column ChiusOggi heading 'Chius|Oggi' format 999.90
column ChiusIeri heading 'Chius|Ieri' format 999.90
column Volume format 999,999,999
```

Come illustra la Figura 14.1, la prima istruzione select estrae semplicemente le azioni delle tre industrie, esegue un semplice calcolo della differenza nei prezzi di chiusura fra oggi e ieri e ordina le azioni per Industria e Società.

Questo è un buon inizio, ma per renderlo più significativo è necessario aggiungere alcune caratteristiche. Viene calcolata una nuova colonna per mostrare la percentuale di variazione fra l'attività di un giorno e quella del successivo:

```
(ChiusOggi/ChiusIeri)*100 - 100 Percento,
```

```

select Industria, Societa,
       ChiusIeri, ChiusOggi,
       (ChiusOggi - ChiusIeri) Nett,
       Volume   from AZIONE
where Industria in ('ELETTRONICA', 'SPAZIO', 'MEDICALE')
order by Industria, Societa
/

```

INDUSTRIA	SOCIETA	Chius Ieri	Chius Oggi	NETT	VOLUME
ELETTRONICA	IDK	95.00	95.25	.25	9,443,523
ELETTRONICA	MEMORY GRAPHICS	15.50	14.25	-1.25	4,557,992
ELETTRONICA	MICRO TOKEN	77.00	76.50	-.50	25,205,667
MEDICALE	AT SPACE	46.75	48.00	1.25	11,398,323
MEDICALE	AUGUST ENTERPRISES	15.00	15.00	.00	12,221,711
MEDICALE	HAYWARD ANTISEPTIC	104.25	106.00	1.75	3,358,561
MEDICALE	KENTGEN BIOPHYSICS	18.25	19.50	1.25	6,636,863
SPAZIO	BRANDON ELLIPSIS	32.75	33.50	.75	25,789,769
SPAZIO	GENERAL ENTROPY	64.25	66.00	1.75	7,598,562
SPAZIO	GENEVA ROCKETRY	22.75	28.00	1.25	22,533,944
SPAZIO	NORTHERN BOREAL	26.75	28.00	1.25	1,348,323
SPAZIO	OCKHAM SYSTEMS	21.50	22.00	.50	7,052,990
SPAZIO	WONDER LABS	5.00	5.00	.00	2,553,712

Figura 14.1 Prezzi di chiusura e volumi di scambio azionario.

Al calcolo è stato assegnato l'alias Percento ed è stata impostata la formattazione per questa colonna. Inoltre, le colonne Società e Industria sono state notevolmente ristrette per fare spazio ad altre colonne che saranno aggiunte tra poco. Naturalmente su un report ampio, per esempio di 132 colonne, questo potrebbe non essere necessario. Qui lo si è fatto per motivi di spazio.

```

column Percento heading 'Percento|Scambio' format 9999.90
column Societa format a8 trunc
column Industria heading 'Ind' format a5 trunc

```

Si esamini il report della Figura 14.2 che include la colonna Percento e la formattazione modificata.

## break on

A questo punto, si configura un comando **break on** per inserire una riga vuota tra la fine di un gruppo Industria e l'inizio del successivo; quindi, viene calcolata la somma del Volume giornaliero per ogni Industria. Si osservi nella Figura 14.2 la coordinazione fra **break on** e **compute sum**. Ora **break on** e **compute sum** vengono estese, come mostrato nella Figura 14.3. L'ordine delle colonne in **break on** è un po' complesso, come verrà spiegato tra poco. **compute sum** è stata incaricata di calcolare il Volume delle colonne Industria e Report. Il volume viene mostrato per ogni industria, come in precedenza, tuttavia ora è stato aggiunto anche un volume totale per tutte

```

break on Industria skip 1

compute sum of Volume on Industria

select Industria,
       Societa,
       ChiusIeri, ChiusOggi,
       (ChiusOggi - ChiusIeri) Nett,
       (ChiusOggi/ChiusIeri)*100 - 100 Percento,
       Volume
  from AZIONE
 where Industria in ('ELETTRONICA', 'SPAZIO', 'MEDICALE')
 order by Industria, Societa
/

```

Ind	SOCIETA	Chius Ieri	Chius Oggi	NETT	Percento Cambio	VOLUME
ELECT	IDK	95.00	95.25	.25	.26	9,443,523
	MEMORY G	15.50	14.25	-1.25	-8.06	4,557,992
	MICRO TO	77.00	76.50	-.50	-.65	25,205,667
<b>*****</b>						
	sum					39,207,182
MEDIC	AT SPACE	46.75	48.00	1.25	2.67	11,398,323
	AUGUST E	15.00	15.00	.00	.00	12,221,711
	HAYWARD	104.25	106.00	1.75	1.68	3,358,561
	KENTGEN	18.25	19.50	1.25	6.85	6,636,863
<b>*****</b>						
	sum					33,615,458
SPAZI	BRANDON	32.75	33.50	.75	2.29	25,789,769
	GENERAL	64.25	66.00	1.75	2.72	7,598,562
	GENEVA R	22.75	27.25	4.50	19.78	22,533,944
	NORTHERN	26.75	28.00	1.25	4.67	1,348,323
	OCKHAM S	21.50	22.00	.50	2.33	7,052,990
	WONDER L	5.00	5.00	.00	.00	2,553,712
<b>*****</b>						
	sum					66,877,300

**Figura 14.2** Report delle azioni con break on Industria e compute sum of Volume.

le industrie. Il comando **compute** è stato esteso per consentire di calcolare la somma su Report. Questo comando dev'essere coordinato con **break on**:

```

break on Report on Industria skip 1
compute sum of Volume on Industria Report

```

Questo codice produce una somma di Volume per l'intero report. Si tratta dei comandi **break on** e **compute** che compaiono nella Figura 14.3. Il posizionamento di **on Report** nel comando **break on** non ha importanza: **on Report** viene sempre calcolato alla fine.

```
break on Report on Industria skip 1
compute sum of Volume on Industria Report
```

```
select Industria,
       Societa,
       ChiusIeri,
       ChiusOggi,
       (ChiusOggi - ChiusIeri) Nett,
       (ChiusOggi/ChiusIeri)*100 - 100 Percento,
       Volume
  from AZIONE
 where Industria in ('ELETTRONICA', 'SPAZIO', 'MEDICALE')
 order by Industria, Societa
 /
```

Portafoglio corrente						1 aprile 1998
Ind	SOCIETA	Industria			Percento	VOLUME
		Chius Ieri	Chius Oggi	NETT		
ELECT IDK	95.00	95.25	.25	.26	9,443,523	
MEMORY G	15.50	14.25	-1.25	-8.06	4,557,992	
MICRO TO	77.00	76.50	-.50	-.65	25,205,667	
*****						
sum					39,207,182	
MEDIC AT SPACE	46.75	48.00	1.25	2.67	11,398,323	
AUGUST E	15.00	15.00	.00	.00	12,221,711	
HAYWARD	104.25	106.00	1.75	1.68	3,358,561	
KENTGEN	18.25	19.50	1.25	6.85	6,636,863	
*****						
sum					33,615,458	
SPAZI BRANDON	32.75	33.50	.75	2.29	25,789,769	
GENERAL	64.25	66.00	1.75	2.72	7,598,562	
GENEVA R	22.75	27.25	4.50	19.78	22,533,944	
NORTHERN	26.75	28.00	1.25	4.67	1,348,323	
OCKHAM S	21.50	22.00	.50	2.33	7,052,990	
WONDER L	5.00	5.00	.00	.00	2,553,712	
*****						
sum					66,877,300	
					139,699,940	

portfoli.sql

**Figura 14.3** Report delle azioni con break on e compute sum estese.

Poi, al report viene assegnato un titolo in alto e uno in fondo, sfruttando la capacità di formattazione estesa di **tttitle** e **btitle**. Per una spiegazione di questi comandi, si faccia riferimento al paragrafo “Comandi di formattazione **tttitle** e **btitle**”.

```
ttitle left 'Portafoglio corrente' -
      right '1 marzo 2002'      skip 1 -
      center 'Industria'   skip 2;

btitle left 'portfoli.sql';
```

## Ordine delle colonne in break on

Se l'ordine delle colonne in break on non è corretto, possono verificarsi dei problemi. Si supponga di voler creare un report delle entrate per società, divisione, dipartimento e progetto (dove una divisione contiene dipartimenti e i dipartimenti contengono i progetti). Se venisse inserito il seguente comando:

```
break on Progetto on Dipartimento on Divisione on Societa
```

i totali di ognuna di queste voci verrebbero calcolati ogni volta che il progetto viene modificato, e inoltre verrebbero accumulati solo i totali per il progetto. Tutto ciò non servirebbe a nulla. Invece, break on dev'essere espressa in ordine dal raggruppamento più grande a quello più piccolo, come mostrato di seguito:

```
break on Societa on Divisione on Dipartimento on Progetto
```

## break on Row

SQLPLUS consente anche di effettuare calcoli (compute) per ogni riga (break on Row). Come per on Report, break on e compute devono essere coordinati.

---

### Comandi di formattazione ttitle e btitle

---

I risultati di questi comandi ttitle e btitle sono visibili nella Figura 14.5, riportata più avanti in questo capitolo.

```
ttitle left 'Portafoglio corrente' -
      right xINDUSTRY      skip 1 -
      center 'Industria'   skip 4;
```

left, right e center definiscono la posizione sulla pagina della stringa che segue. Un trattino alla fine della riga indica che segue un'altra riga di comandi di titolo. skip comunica quante righe vuote devono essere stampate dopo questa riga. Il testo racchiuso fra apici singoli viene stampato così com'è. Le parole non racchiuse fra apici singoli in genere sono variabili; se sono state definite da accept, NEW\_VALUE o define, nel titolo verranno stampati i loro valori. Se non sono state definite, ne verrà stampato il nome.

```
btitle left 'portfoli.sql' on ' xOGGI -
      right 'Pagina ' format 999 sql.pno;
```

sql.pno è una variabile che contiene il numero della pagina corrente. I comandi di formattazione possono essere posizionati ovunque nel titolo e controllano la formattazione di ogni variabile seguente in ttitle o in btitle, finché non si incontra un altro comando di formato.

Per altre opzioni di questi comandi, si può consultare la voce ttitle nella guida di riferimento alfabetica del Capitolo 42.

## Aggiunta di viste

Perché il report sia utile, è importante effettuare i calcoli di ogni azione in relazione al suo settore industriale e al complesso. Perciò vengono create due viste, mostrate di seguito. La prima riassume il Volume delle azioni, raggruppate per Industria. La seconda riassume il volume totale delle azioni.

```
create or replace view INDUSTRIA as
select Industria, SUM(Volume) Volume
  from AZIONE
 where Industria in ('ELETTRONICA', 'SPAZIO', 'MEDICALE')
group by Industria
/
create or replace view MERCATO as
select SUM(Volume) Volume
  from AZIONE
 where Industria in ('ELETTRONICA', 'SPAZIO', 'MEDICALE')
/
```

Questa pratica di creare viste in un report di SQLPLUS è adatta solo per viste che vengono utilizzate più diffusamente. Per le viste monouso si possono creare viste in linea nella clausola `from` per raggiungere i propri scopi (si faccia riferimento al Capitolo 11).

## Colonne utilizzate con ttitle e btitle

Ora vengono aggiunte alcune definizioni di colonne, insieme a una nuova colonna, PartDiInd, (spiegata più avanti in questo capitolo). Si osservi il punto A nella Figura 14.4.

Due colonne che compariranno in `ttitle` e `btitle`, vengono inserite con il comando `NUOVO_VALORE`. Si analizzino le definizioni nel punto B della Figura 14.4 e gli effetti nel punto 1 della Figura 14.5. La colonna Oggi è stata utilizzata per riportare la data odierna in `btitle`. Come funziona? Innanzi tutto, Oggi è un alias della colonna `SysDate` formattata nell'istruzione `select`:

```
TO_CHAR(SysDate,'fmMonth ddth, yyyy') Oggi
```

`NUOVO_VALORE` inserisce il contenuto della colonna Oggi (in questo caso, 1 marzo 2002) in una variabile chiamata `xOggi`, che viene poi utilizzata in `btitle`:

```
btitle left 'portfoli.sql on ' xOggi -
      right 'Pag ' format 999 sql.pno;
```

La variabile potrebbe avere qualsiasi nome; è stato scelto `xOggi` per poterla individuare facilmente nel listato, ma avrebbe potuto avere persino lo stesso nome della colonna Oggi, oppure qualcos'altro come `VarData` o `XYZ`. `portfoli.sql` è semplicemente il nome del file di avvio utilizzato per produrre questo report. È utile stampare il nome del file di avvio utilizzato per creare un report in qualche punto del report stesso, in modo che, qualora si presenti la necessità di eseguirlo ancora, sia possibile trovarlo facilmente.

L'ultima parte di `format 999 sql.pno`, `sql.pno`, è una variabile che contiene sempre il numero della pagina corrente. È possibile utilizzarla ovunque sia in `ttitle` che in `btitle`. Se la si inserisce in `btitle` dopo la parola "right", appare nell'angolo in basso a destra di ogni pagina.

```

column User noprint
column PartDiInd heading 'Parte|di Ind' format 999.90 ----- A
column Oggi    NEW_VALUE xOGGI noprint format a1 trunc
column Industria NEW_VALUE xINDUSTRIA ----- B
ttitle left 'Portfolio corrente' -
          right xINDUSTRIA           skip 1 -
          center 'Industria'      skip 4
btitle left 'portfoli.sql on ' xOGGI -
          right 'Pag ' format 999 sql.pno

clear breaks ----- C
clear computes

break on Report page on Industria page ----- D

compute sum of Volume on Report Industria
compute sum of PartDiInd on Industria
compute avg of Nett Percento on Industria
compute avg of Nett Percento PartDiInd on Report ----- E

select S.Industria,
       Societa,
       ChiusIeri, ChiusOggi,
       (ChiusOggi - ChiusIeri) Nett,
       (ChiusOggi - ChiusIeri)*(S. Volume/I.Volume) PartDiInd,
       (ChiusOggi/ChiusIeri)*100 - 100 Percento,
       S.Volume,
       TO_CHAR(SysDate,'dd fmMonth yyyy') Oggi
  from AZIONE S, INDUSTRIA I
 where S.Industria = I.Industria
   AND I.Industria in ('ELETTRONICA', 'SPAZIO', 'MEDICALE')
  order by I.Industria, Societa
/

```

**Figura 14.4** Report delle azioni con break on e compute sum estese.

format 999, che precede questa variabile, determina il formato per il numero di pagina. Un comando di formattazione come questo, ogni volta che è presente in ttitle o btitle, definisce il formato di qualsiasi numero o carattere nelle variabili che seguono, fino al termine del comando ttitle o btitle, a meno che non sia presente un altro comando di formato.

### **Un avvertimento sulle variabili**

Esistono altri due elementi importanti nel comando column:

```
column Oggi NUOVO_VALORE xOggi noprint format a1 trunc
```

noprint comunica a SQLPLUS di non visualizzare questa colonna durante la stampa dei risultati dell'istruzione SQL. Senza questa opzione, la data apparirebbe in ogni riga del report. format a1 trunc è un po' più complesso. Le date che sono state riformattate con TO\_CHAR assumono una

Portafoglio corrente
(2)
ELETTRONICA

Industria

Ind	SOCIETA	Chius Ieri	Chius Oggi	NETT	Parte di Ind	Percento Cambio	VOLUME
ELETT	IDK	95.00	95.25	.25	.06	.26	9,443,523
	MEMORY G	15.50	14.25	-1.25	-.15	-8.06	4,557,992
	MICRO TO	77.00	76.50	-.50	-.32	-.65	25,205,667
*****							
avg				-.50		-2.82	
sum					-.41		39,207,182

portfoli.sql on March 1st, 2002 —— (1) Pag 1

Portafoglio corrente
(3)
MEDICALE

Industria

Ind	SOCIETA	Chius Ieri	Chius Oggi	NETT	Parte di Ind	Percento Cambio	VOLUME
MEDIC	AT SPACE	46.75	48.00	1.25	.42	2.67	11,398,323
	AUGUST E	15.00	15.00	.00	.00	.00	12,221,711
	HAYWARD	104.25	106.00	1.75	.17	1.68	3,358,561
	KENTGEN	18.25	19.50	1.25	.25	6.85	6,636,863
*****							
avg				1.06		2.80	
sum					.85		33,615,458

portfoli.sql on March 1st, 2002 —— (2) Pag 2

**Figura 14.5** Report per industria, con un'industria per pagina. (segue)

larghezza predefinita di circa 100 caratteri (come è stato spiegato nel Capitolo 9). Anche se l'opzione `noprint` è attiva, SQLPLUS rimane confuso e conta comunque la larghezza della colonna `Oggi` per stabilire se è stata superata la dimensione `linesize`. La conseguenza è che viene completamente distrutta la formattazione delle righe in visualizzazione o in stampa. Cambiando il formato in `a1 trunc`, questo effetto viene ridotto al minimo.

L'altra colonna con `NUOVO_VALORE` è `Industria`. Questa, in effetti, ha già alcune definizioni di colonna:

```
column Industria heading 'Ind' format a5 trunc
```

e ora ne viene aggiunta una nuova, come mostrato nel listato seguente (si ricordi che istruzioni diverse per una colonna possono essere fornite in diverse righe).

Portafoglio corrente

Industria

Ind	SOCIETA	Chius Ieri	Chius Oggi	NETT	Parte di Ind	Percento Cambio	VOLUME
SPAZI	BRANDON	32.75	33.50	.75	.29	2.29	25,789,769
	GENERAL	64.25	66.00	1.75	.20	2.72	7,598,562
	GENEVA R	22.75	27.25	4.50	1.52	19.78	22,533,944
	NORTHERN	26.75	28.00	1.25	.03	4.67	1,348,323
	OCKHAM S	21.50	22.00	.50	.05	2.33	7,052,990
	WONDER L	5.00	5.00	.00	.00	.00	2,553,712
<b>*****</b>							
avg				1.46		5.30	
sum					2.08		66,877,300

portfoli.sql on March 1st, 2002

SPAZIO

Portafoglio corrente

Industria

Ind	SOCIETA	Chius Ieri	Chius Oggi	NETT	Parte di Ind	Percento Cambio	VOLUME
				.88	.19	2.66	139,699,940

portfoli.sql on March 1st, 2002

SPAZIO

Figura 14.5 Report per industria, con un'industria per pagina. (*continua*)

```
column Industria NUOVO_VALORE xIndustria
```

Come per Oggi, questo comando column inserisce il contenuto della colonna dall'istruzione select in una variabile chiamata xIndustria, il cui valore è coordinato con break on. Essa ottiene il valore di Industria quando viene letta la *prima* riga e lo conserva finché non viene incontrato un nuovo valore e break on non fa passare a una nuova pagina, come mostrato nei punti 2, 3 e 4 della Figura 14.5.

Di seguito sono descritte le azioni effettuate da SQLPLUS:

1. Ritarda la stampa di qualsiasi elemento su una pagina finché break on non rileva un cambiamento nel valore di Industria o finché non sono state recuperate tante righe sufficienti a riempire la pagina.
2. Stampa tttitle con il valore di xIndustria prima della modifica (o prima che la pagina sia stata riempita).

3. Sposta il valore di xIndustria in un variabile “VECCHIO\_VALORE” e lo salva.
4. Stampa le righe sulla pagina.
5. Carica il nuovo valore in xIndustria.
6. Stampa btitle.
7. Inizia a raccogliere le righe per la pagina successiva e riparte dal primo passaggio.

Questo significa che, se xIndustria fosse stata posta in btitle invece che in ttitle, avrebbe contenuto il valore di Industria della pagina seguente e non di quella che viene stampata. MEDICALE (punto 3) sarebbe in fondo alla pagina 1, SPAZIO (punto 4) in fondo alla pagina 2 e così via. Per utilizzare btitle in modo appropriato rispetto ai valori delle colonne restituite nella query, è consigliabile impiegare l’istruzione seguente:

```
column Industria VECCHIO_VALORE xIndustria
```

## **Ulteriori informazioni su break on e compute**

Il punto C nella Figura 14.4 precede immediatamente i comandi break on e compute. Quando un comando compute entra in azione, continua a essere attivo finché non viene rimosso o non si esce da SQLPLUS. Questo significa che si possono avere comandi compute dei report precedenti che vengono eseguiti su quello corrente, producendo effetti indesiderati di qualsiasi tipo.

Anche il comando break on rimane, anche se viene completamente sostituito da uno nuovo eventualmente presente. È buona norma inserire i comandi clear breaks e clear computes immediatamente prima di configurare nuovi comandi break on e compute.

Ecco le opzioni disponibili per break on:

- break on colonna
- break on row
- break on report

*colonna* può essere un nome di colonna, un alias o un’espressione come SUBSTR(Industria,1,4). Ciascuna di queste opzioni può essere seguita da una delle azioni seguenti:

- skip *n*
- skip page

oppure da niente. break on *colonna* produce l’azione ogni volta che il valore nella colonna selezionata cambia. break on row produce un’interruzione a ogni riga. break on report intraprende l’azione specificata ogni volta che un report viene terminato.

Le azioni skip servono per saltare una o più linee (ovvero, stampare delle righe vuote) o per passare all’inizio di una nuova pagina. Si ricordi (dal Capitolo 3) che break on viene emesso soltanto una volta, con tutte le colonne e le azioni desiderate. Si veda il punto D nella Figura 14.4.

Il comando compute invece può essere riutilizzato per ogni tipo di calcolo e per una o più colonne contemporaneamente. Il punto E mostra una varietà di modi in cui può essere impiegato. Si osservi come le colonne compaiano in entrambi i lati di on nei comandi compute. Inoltre, nei comandi break on o compute non viene utilizzata alcuna virgola.

Di seguito sono riportati i possibili calcoli:

compute avg	compute num
compute count	compute sum
compute max	compute std
compute min	compute var

Questi comandi hanno, per una colonna in un report di SQLPLUS, lo stesso significato che AVG(), COUNT(), MAX(), MIN(), STDDEV(), SUM() e VARIANCE() hanno in SQL. Tutti, eccetto num, ignorano NULL e nessuno è in grado di utilizzare la parola chiave DISTINCT. compute num è simile a compute count, con la differenza che il secondo conta solo le righe non nulle, mentre il primo le conta *tutte*.

### **Visualizzazione di break e compute correnti**

Se si inserisce solo la parola break o compute in SQLPLUS, vengono visualizzati tutti i break e compute che al momento sono attivi.

### **Esecuzione contemporanea di diversi compute**

La Figura 14.4 contiene le seguenti istruzioni compute:

```
compute sum of Volume on Report Industria
compute sum of PartDiInd on Industria
compute avg of Nett Percento on Industria
compute avg of Nett Percento PartDiInd on Report
```

Se si esaminano i punti 5 e 6 nella Figura 14.5, si vedono i loro effetti. Si osservi che il calcolo avg per ogni colonna appare su una riga, mentre sum (dove è presente) appare sulla riga seguente. Inoltre, le parole sum e avg compaiono *sotto la colonna* che segue la parola on nell'istruzione compute. Esse mancano al punto 6 per il calcolo dei totali generali perché sono calcolati su Report, che non è una colonna, per cui sum e avg non possono comparire sotto una colonna.

La somma dei Volumi al punto 5 riguarda solamente le azioni dell'industria Spazio. La somma dei Volumi al punto 6 è calcolata per tutte le industrie (la pagina 4 di questo report contiene soltanto totali e medie generali). Si osservi che il primo compute serve per la somma di una colonna, Volume, sia su Report sia su Industria (tra l'altro, l'ordine delle colonne in compute è irrilevante, diversamente da quanto avviene per break on).

Il compute successivo, per PartDiInd su Industria, richiede alcune spiegazioni (PartDiInd si riferisce alla parte di variazione dell'industria o di cambiamento nei prezzi delle azioni rispetto a tutte le azioni dell'industria). PartDiInd proviene da una porzione dell'istruzione select:

```
(ChiusOggi - ChiusIeri)*(S.Volume/I.Volume) PartDiInd,
```

Questo è un calcolo che pondera la variazione netta in un'azione rispetto alle altre della sua Industria, in base al volume d'affari. È opportuno confrontare la variazione netta (Net) effettiva di BRANDON e di NORTHERN nell'industria SPAZIO. BRANDON è variata solo di 0.75, ma con una quantità trattata che superava i 25 milioni. NORTHERN è variata di 1.25, ma con una quantità trattata di un milione circa. Il contributo di BRANDON alla variazione positiva nell'industria SPAZIO è considerevolmente maggiore di quello apportato da NORTHERN; questo è visibile nei valori relativi alle due aziende sotto la colonna PartDiInd.

Ora, si mettano a confronto la somma (sum) della colonna PartDiInd, 2.08, con la media (avg) della colonna Netta, 1.46 (dall'istruzione compute immediatamente successiva). Qual è più rappresentativa della variazione dei prezzi delle azioni in questa industria? La somma di PartDiInd, perché i suoi valori sono ponderati con il volume delle azioni. Questo esempio è stato presentato per mostrare la differenza tra la semplice media in una colonna e la media ponderata, e per illustrare come vengono calcolate informazioni di riepilogo. In un caso vengono calcolate eseguendo la media; in un altro eseguendo la somma.

Non è questa la sede per affrontare una trattazione dettagliata di medie ponderate, percentuali o calcoli statistici, tuttavia è opportuno evidenziare le differenze significative che risultano

dai vari metodi di calcolo. Questo spesso influenza sulle decisioni da prendere, perciò servono attenzione e prudenza.

L'ultimo **compute** calcola le medie di Nett, Percento e PartDiInd su Report. Questi valori sono mostrati al punto 6 rispettivamente come 0.88, 0.19 e 2.66. Si osservi l'ultima linea della Figura 14.6. Questo è lo stesso report della Figura 14.5, con alcune eccezioni. Occupa una pagina invece che quattro; nel titolo superiore a destra si trova la data, non il settore industriale e la riga in fondo ha un risultato aggiuntivo e diverso per le medie e i totali:

Medie e totali di mercato	1.09	2.66	139,699,940
---------------------------	------	------	-------------

Questi sono i risultati corretti, diversi da quelli che sono o possono essere calcolati dalle colonne visualizzate utilizzando *qualsiasi* istruzione **compute**, perché nelle istruzioni **compute** manca la ponderazione del volume totale dell'industria. Quindi, come si è arrivati a questo risultato? La risposta è fornita nella Figura 14.7; al punto F, **btitle** è stato disattivato e al punto G (subito dopo l'istruzione **select** che ha prodotto il corpo principale del report) si trova una **select**

Portafoglio corrente							1 April 1998
Industria							
Ind	SOCIETA	Chius Ieri	Chius Oggi	NETT	Parte di Ind	Percento Cambio	VOLUME
ELETT	IDK	95.00	95.25	.25	.06	.26	9,443,523
	MEMORY G	15.50	14.25	-1.25	-.15	-8.06	4,557,992
	MICRO TO	77.00	76.50	-.50	-.32	-.65	25,205,667
<b>*****</b>							
avg				-.50		-2.82	
sum					-.41		39,207,182
MEDIC	AT SPACE	46.75	48.00	1.25	.42	2.67	11,398,323
	AUGUST E	15.00	15.00	.00	.00	.00	12,221,711
	HAYWARD	104.25	106.00	1.75	.17	1.68	3,358,561
	KENTGEN	18.25	19.50	1.25	.25	6.85	6,636,863
<b>*****</b>							
avg				1.06		2.80	
sum					.85		33,615,458
SPAZI	BRANDON	32.75	33.50	.75	.29	2.29	25,789,769
	GENERAL	64.25	66.00	1.75	.20	2.72	7,598,562
	GENEVA R	22.75	27.25	4.50	1.52	19.78	22,533,944
	NORTHERN	26.75	28.00	1.25	.03	4.67	1,348,323
	OCKHAM S	21.50	22.00	.50	.05	2.33	7,052,990
	WONDER L	5.00	5.00	.00	.00	.00	2,553,712
<b>*****</b>							
avg				1.46		5.30	
sum					2.08		66,877,300
				.88	.19	2.66	
							139,699,940
  Medie e totali mercato							
1.09							
2.66							
139,699,940							

**Figura 14.6** Report rivisto, con le medie e i totali corretti del mercato azionario.

aggiuntiva che include l'etichetta “Medie e totali di mercato” e un calcolo appropriato delle medie e dei totali.

Questa istruzione select è preceduta da set heading off e ttitle off. Questi comandi sono necessari perché una nuova select normalmente fa stampare un nuovo titolo in alto. Il comando

```

column User noprint
column Today      NEW_VALUE xTODAY noprint format al trunc
column Industria NEW_VALUE xINDUSTRIA
column Nett format 99.90
column Industria format a11
column Societa format a18
column ChiusOggi heading 'Chius|Oggi' format 999.90
column ChiusIeri heading 'Chius|Ieri.' format 999.90
column Volume format 999.999.999
column Percento heading 'Percento|Cambio' format 9999.90
column Societa format a8 trunc
column Industria heading 'Ind' format a5 trunc

ttitle left 'Portafoglio corrente' right xOGGI skip 1 -
           center 'Industria'      skip 4;

btitle off —————— F

clear breaks
clear computes

break on Report on Industria skip 1

compute sum of Volume on Report Industria
compute sum of PartDiInd on Industria
compute avg of Nett Percento on Industria
compute avg of Nett Percento PartDiInd on Report

select S.Industria,
       Societa,
       ChiusIeri, ChiusOggi,
       (ChiusOggi - ChiusIeri) Nett,
       (ChiusOggi - ChiusIeri)*(S.Volume/I.Volume) PartDiInd,
       (ChiusOggi/ChiusIeri)*100 - 100 Percento,
       S.Volume, User,
       To CHAR(SysDate,'ddth fmMonth yyyy')Oggi
  from AZIONE S, INDUSTRIA I
 where S.Industria = I.Industria
   AND I.Industria in ('ELETTRONICA', 'SPAZIO', 'MEDICALE')
 order by I.Industria, Societa
/
set heading off
ttitle off
select 'Medie e totali mercato' ,
       SUM(((ChiusOggi-ChiusIeri)*S.Volume)/M.Volume) Nett, —————— G
       AVG((ChiusOggi/ChiusIeri))*100 - 100 Percento,
       SUM(S.Volume) Volume
  from AZIONE S, MERCATO M
 where Industria in ('ELETTRONICA', 'SPAZIO', 'MEDICALE')

```

**Figura 14.7** Comandi per la produzione di medie e totali corretti del mercato azionario.

set heading off serve per disattivare i titoli delle *colonne*, che normalmente verrebbero visualizzati sopra questa linea. È stato necessario disattivare btitle al punto F prima che venisse eseguita la prima istruzione select, perché per completare questa istruzione sarebbe stato stampato btitle prima di iniziare l'esecuzione della seconda select.

Come mostrato nell'output dei report di esempio in questo capitolo (Figure 14.2, 14.3, 14.5 e 14.6), quando viene utilizzato un comando compute, al valore calcolato viene assegnata un'etichetta con la funzione che è stata eseguita. Per esempio, nella Figura 14.6, sono state calcolate le funzioni AVG e SUM; l'output mostra un'etichetta "avg" per il calcolo di AVG e una "sum" per il calcolo di SUM. È possibile ridefinire le etichette predefinite e specificarne alcune personalizzate per le funzioni calcolate nel comando compute. Per ulteriori dettagli, si può consultare la clausola label di COMPUTE nel Capitolo 42. Per evitare che i comandi compute vengano applicati alla colonna Volume della seconda query, il comando clear computes viene eseguito dopo la prima query.

## 14.2 set termout off e set termout on

Un'altra coppia utile di comandi è quella costituita da set termout off e set termout on. Il primo viene utilizzato spesso in un file di avvio immediatamente prima del comando spool, e il secondo subito dopo. L'effetto è quello di eliminare la visualizzazione del report sullo schermo. Nella stampa dei report questo fa risparmiare tempo (infatti, verranno eseguiti più velocemente), ed evita l'irritante scorrere dei dati sullo schermo. La registrazione su un file continua a funzionare correttamente.

## 14.3 Variabili in SQLPLUS

Se si utilizza SQLPLUS in modo interattivo, si possono controllare i valori correnti di ttitle e btitle in qualsiasi momento digitando i loro nomi da soli su una riga. SQLPLUS mostra immediatamente il loro contenuto:

```
ttitle
ttitle OFF and is the following 92 characters:

left 'Portafoglio corrente'      right xOGGI
skip 1           center 'Industria' skip 2
```

Si noti la presenza di xOGGI, che è una variabile, invece del suo contenuto corrente. SQLPLUS è in grado di memorizzare e utilizzare molte variabili, e quelle impiegate in ttitle e btitle sono solo alcune. Le variabili correnti e i loro valori possono essere visualizzati con define:

```
define
DEFINE _SQLPLUS_RELEASE = "900010000" (CHAR)
DEFINE _EDITOR          = "Notepad" (CHAR)
DEFINE _O_VERSION        = "Oracle9i Enterprise Edition Release 9.0.1.0.0
With the Partitioning option
JServer Release 9.0.1.0.0 -Beta" (CHAR)
DEFINE _O_RELEASE         = "900010000" (CHAR)
DEFINE XINDUSTRIA        = "SPAZIO" (CHAR)
DEFINE XOGGI              = "10 April 2002" (CHAR)
```

La variabile \_EDITOR indica l'editor che viene utilizzato quando si digita la parola edit. Nel Capitolo 6 si è mostrato come configurare questa variabile nel file login.sql. Le altre variabili identificano la versione e la release di Oracle utilizzate.

Le ultime variabili sono state definite nelle query del mercato azionario e i loro contenuti sono proprio quelli visualizzati nei report. SQLPLUS memorizza le variabili per ttitle e btitle impostandole a un certo valore, poi consente di utilizzarle ogniqualvolta viene eseguita una query. In effetti, le variabili possono essere impiegate in molte parti di un report, oltre che nei titoli.

Si supponga che il database di azioni venga aggiornato automaticamente da un servizio di determinazione del prezzo, e che si desideri controllare regolarmente i prezzi di chiusura e i volumi su una base azione per azione. Questo può essere ottenuto facilmente con delle variabili nella clausola where. Normalmente si scriverebbe questa query:

```
select Societa, ChiusIeri, ChiusOggi, Volume
  from STOCK where Societa = 'IDK';
```

SOCIETA	CHIUS OGGI	CHIUS IERI	VOLUME
IDK	95.00	95.25	9,443,523

In alternativa, la si potrebbe scrivere in un file di avvio chiamato, per esempio, chiusura.sql:

```
set heading on
column ChiusOggi heading 'Chius|Oggi' format 999.90
column ChiusIeri heading 'Chius|Ieri' format 999.90
column Volume format 999,999,999
column Societa format a20

accept xSocieta prompt 'Inserire nome Societa'

select Societa, ChiusIeri, ChiusOggi, Volume
  from STOCK
 where Societa = '&xSocieta';
```

In questo file, xSocietà è una variabile inventata, come xIndustria o xOggi. accept comunica a SQLPLUS di accettare l'input dalla tastiera, e prompt di visualizzare un messaggio. Si noti che il nome della variabile, quando si trova nell'istruzione SQL select, dev'essere preceduto da una "e commerciale" (&), ma non quando si trova in accept o in un titolo. Poi, quando si digita:

```
start closing.sql
```

lo schermo mostra questo:

```
Inserire nome Societa:
```

Quindi, si digita MEMORY GRAPHICS. SQLPLUS visualizzerà quanto segue:

```
select Societa, ChiusIeri, ChiusOggi, Volume
  from AZIONE
 where Societa = '&xSocieta';

old  3: where Societa = '&xSocieta'
new  3: where Societa = 'MEMORY GRAPHICS'
```

	Chius Ieri.	Chius Oggi	VOLUME
----- MEMORY GRAPHICS	15.50	14.25	4,557,992

Innanzi tutto, viene mostrata la query che è stata configurata. Successivamente, la clausola `where`, prima con la variabile, poi con il valore digitato. Infine, i risultati della query.

Se poi si digita `start closing.sql` di nuovo, ma con IDK come nome per Società, i valori old e new mostrati sono i seguenti:

```
old  3: where Societa = '&xSocieta'
new  3: where Societa = 'IDK'
```

e il risultato riguarda IDK. Non è necessario che i comandi `select`, `old` e `new` vengano mostrati ogni volta: possono essere controllati, il primo con il comando `set echo` e gli altri due con `set verify`. Per vedere come sono impostati attualmente questi comandi, occorre digitare il comando `show` seguito dalla parola chiave `verify` o `echo`.

```
show verify
verify ON
```

```
show echo
echo OFF
```

La versione rivista del file di avvio è la seguente:

```
column ChiusOggi heading 'Chius|Oggi' format 999.90
column ChiusIeri heading 'Chius|Ieri' format 999.90
column Volume format 999,999,999
column Societa format a20
set echo off
set verify off
set sqlcase upper
accept xSocieta prompt 'Inserire nome Societa: '

select Societa, ChiusIeri, ChiusOggi, Volume
  from AZIONE
 where Societa = '&xSocieta';
```

`set sqlcase upper` comunica a SQLPLUS di convertire in maiuscolo, prima di eseguire la query, qualunque cosa venga immessa nella variabile (utilizzando `accept`). Questo può essere utile se i dati sono stati memorizzati in lettere maiuscole (una buona norma) e non si vuole indurre le persone a digitare necessariamente in maiuscolo, quando eseguono una query. Ora, quando il file di avvio viene eseguito, accade quanto segue:

```
start closing.sql

Inserire nome Societa: memory graphics

-----  


|                          | Chius<br>Ieri. | Chius<br>Oggi | VOLUME    |
|--------------------------|----------------|---------------|-----------|
| -----<br>MEMORY GRAPHICS | 15.50          | 14.25         | 4,557,992 |


```

L'uso di variabili nei file di avvio può essere molto utile, in particolare per report in cui il formato base rimane immutato mentre alcuni parametri cambiano, come data, divisione del-

l'azienda, nome dell'azione, progetto, cliente e così via. Quando viene avviato un report, vengono richiesti all'utente questi particolari e poi viene iniziata l'esecuzione. Come è stato mostrato in precedenza, digitando solo la parola **define** si ottiene un elenco di tutte le variabili attualmente definite. Digitando **define** con il nome di una sola variabile si visualizza il contenuto di quest'ultima:

```
define xSocieta
DEFINE XSOCIETA      = "memory graphics"  (CHAR)
```

Dopo aver completato un file di avvio, qualsiasi variabile viene conservata da SQLPLUS finché non si esce o non si decide di rimuovere le variabili intenzionalmente con **undefine**:

```
undefine xSocieta
```

Se ora si cerca di vedere il valore della variabile si ottiene quanto segue:

```
define xSocieta
SP2-0135: symbol xsocieta is UNDEFINED
```

Si può anche definire una variabile nel file di avvio senza impiegare il comando **accept**. È sufficiente assegnarle un valore, come mostrato di seguito:

```
define xSocieta = 'IDK'
```

In qualsiasi punto del file di avvio compaia **xSocietà**, viene sostituita da 'IDK'.

## **Altri luoghi dove utilizzare le variabili**

Qualsiasi variabile definita utilizzando **accept** o **define** può essere impiegata direttamente in un comando **btitle** o **ttitle** senza utilizzare il comando di colonna **NUOVO\_VALORE**. **NUOVO\_VALORE** prende semplicemente il contenuto di una colonna ed esegue il proprio comando **define** per la variabile che lo segue. La sola differenza nell'utilizzare la variabile nel titolo, invece che nell'istruzione SQL, è che nel titolo la variabile non è preceduta dal simbolo &.

Le variabili possono essere impiegate anche in un file di avvio di configurazione, come viene spiegato più avanti nel capitolo nel paragrafo "Uso di mask.sql".

### **14.4 Formattazione di numeri**

Il metodo predefinito utilizzato da SQLPLUS per formattare i numeri consiste semplicemente nel giustificarli a destra in una colonna, senza virgolette, utilizzando il punto decimale solo se il numero non è un intero. La Tabella TESTNUMERO contiene le colonne Valore1 e Valore2, che hanno valori identici. Queste colonne vengono usate per mostrare come funziona la formattazione di numeri. La prima query mostra la formattazione predefinita:

```
select Valore1, Valore2 from TESTNUMERO;
```

VALORE1	VALORE2
0	0
.0001	.0001

1234	1234
1234.5	1234.5
1234.56	1234.56
1234.567	1234.567
98761234.6	98761234.6

La quinta riga è NULL. Si osservi come la posizione del punto decimale cambi da una riga all'altra. Dal momento che le colonne possono essere formattate individualmente nelle query, il formato predefinito può essere modificato:

```
set numformat 9,999,999
select Valore1, Valore2 from TESTNUMERO;
```

VALORE1	VALORE2
0	0
0	0
1.234	1,234
1.235	1,235
1.235	1.235
1.235	1.235
#####	#####

Ora l'ottava riga è riempita con segni di cancelletto (#). Il problema di questa riga è che il formato definito è troppo stretto. Si può rimediare aggiungendo un'altra cifra a sinistra, come mostrato di seguito:

```
set numformat 99,999,999
select Valore1, Valore2 from TESTNUMERO;
```

VALORE1	VALORE2
0	0
0	0
1.234	1,234
1.235	1,235
1.235	1.235
1.235	1.235
98.761,235	98,761,235

## 14.5 Uso di mask.sql

Potrebbe essere necessario riutilizzare regolarmente le colonne di un report in vari altri report. Invece di riscrivere i comandi di formattazione per queste colonne in ogni file di avvio, può essere utile conservare tutti i comandi di colonna fondamentali in un singolo file. Questo file potrebbe prendere il nome di mask.sql, perché contiene le informazioni di formattazione (o “maschera”) per le colonne. Per esempio, il file potrebbe essere come il seguente:

```

REM      File mask.sql
set numwidth 12
set numformat 999,999,999.90

column Nett format 99.90
column Industria format a11
column Societa format a18
column ChiusOggi heading 'Chius|Oggi' format 999.90
column ChiusIeri heading 'Chius|Ieri' format 999.90
column Volume format 999,999,999

define xDipartimento = 'Dip. 3404 Pianificazione Sistema'

```

Questo viene poi incorporato in un file di avvio (in genere all'inizio) semplicemente inserendo questa linea:

```
start mask.sql
```

## 14.6 show all e spool

Sono stati trattati vari comandi che utilizzano il comando set, e per i quali è possibile determinare lo stato corrente con il comando show, come feedback, echo, verify, heading e così via. Esistono circa 50 comandi che possono essere impostati Ed è possibile visualizzarli tutti contemporaneamente con:

```
show all
```

Sfortunatamente, i comandi scorrono così velocemente sullo schermo che risulta impossibile leggerli tutti. Si può risolvere questo problema con lo spooling. È sufficiente digitare il comando spool seguito dal nome di un file, come nell'esempio precedente, eseguire il comando show all e poi spool off. Si può poi esaminare lo stato corrente di tutti i comandi set utilizzando il proprio editor sul file. È possibile trovare questi comandi nel Capitolo 42 alla fine del libro, sotto la voce SET. Le righe restituite dal comando show all sono in ordine alfabetico.

In alternativa, si può usare il comando store, descritto nel Capitolo 6, per salvare le impostazioni ambientali correnti di SQLPLUS in un file (come mask.sql). Per ripristinare queste impostazioni ambientali si potrà eseguire il file più avanti.

## 14.7 Aggiunta di linee vuote

Le informazioni recuperate da un database spesso vanno bene così come sono, con una riga per i titoli di colonna e le colonne dei dati disposte sotto di essa. In alcune occasioni però, è preferibile un layout diverso. Ogni tanto, questo layout può essere realizzato attraverso colonne letterali con spazi vuoti, per posizionare in modo appropriato i dati reali su più di una linea e allinearli correttamente. A queste colonne letterali viene assegnato un alias in SQL e un'intestazione vuota nel comando column. La tecnica ricalca quella descritta per la linea dei totali nella Figura 14.6. Per esempio, si esamini questo listato:

```

column Industria format a14 trunc
column B format a21 heading ''
column Societa format a20
column Volume format 999,999,999 justify left
set linesize 60

```

```

select Industria, '' B,
       Societa,
       ChiusIeri, ChiusOggi,
       (ChiusOggi - ChiusIeri) Nett,
       Volume
  from AZIONE
 where Industria in ('PUBBLICITA','VESTIARIO')
 order by Industria, Societa;

```

INDUSTRIA	SOCIETA		
CHIUSIERI	CHIUSOGGI	NETT	VOLUME
PUBBLICITA			AD SPECIALTY
31.75	31.75	.00	18,333,876
			MBK COMMUNICATIONS
43.25	41.00	-2.25	10,022,980
			NANCY LEE FEATURES
13.50	14.25	.75	14,222,692
VESTIARIO			ROBERT JAMES APPAREL
23.25	24.0	.75	19,032,481

Una colonna letterale di uno spazio, con un alias B, è stata definita dal comando column con una larghezza di 21 caratteri e con un'intestazione vuota. Essa viene utilizzata proprio per spostare i nomi delle società in modo che le colonne siano allineate come si desidera, con il Volume delle azioni direttamente sotto il nome della Società, e con la colonna B per ChiusOggi e Nett allineata con gli spazi vuoti.

### **fold\_after e fold\_before**

È opportuno esaminare come il comando di colonna fold\_after agisce su ogni colonna nell'istruzione select. In pratica, questo comando inserisce le colonne nelle proprie linee.

```

clear columns
column Nett format 99.90
column A format a15 fold_after 1
column Societa format a20 fold_after 1
column ChiusOggi heading 'Chius|Oggi' format 999.90 -
    fold_after
column ChiusIeri heading 'Chius|Ieri' format 999.90 -
    fold_after 1
column Nett fold_after 1
column Volume format 999,999,999 fold_after 1
set heading off
select Industria||',' A,
       Societa,

```

```
ChiusIeri,  
ChiusOggi, (ChiusOggi - ChiusIeri) Nett,  
Volume  
from AZIONE  
where Industria in ('PUBBLICITA','VESTIARIO')  
order by Industria, Societa;
```

La query precedente produce questo output. Si osservi che, anche se dopo la clausola `fold_after` di `ChiusOggi` non c'erano numeri, SQLPLUS ha utilizzato un valore predefinito pari a 1.

```
PUBBLICITA,  
AD SPECIALTY  
31.75  
31.75  
.00  
18,333,876
```

```
PUBBLICITA.  
MBK COMMUNICATIONS  
43.25  
41.00  
-2.25  
10,022,980
```

```
PUBBLICITA,  
NANCY LEE FEATURES  
13.50  
14.25  
.75  
14,222,692
```

```
VESTIARIO.  
ROBERT JAMES APPAREL  
23.25  
24.00  
.75  
19,032,481
```

## 14.8 Ulteriori controlli per la realizzazione di report

Molti dei comandi illustrati in questo come in altri capitoli, hanno diverse opzioni oltre a quelle utilizzate in questi esempi. Tutte le opzioni per ciascuno dei comandi sono disponibili nel Capitolo 42, sotto il nome dei singoli comandi.



## Capitolo 15

# Modifica dei dati: insert, update, merge e delete

- 15.1 **insert**
- 15.2 **rollback, commit e autocommit**
- 15.3 **Inserimenti in più tabelle**
- 15.4 **delete**
- 15.5 **update**
- 15.6 **Uso del comando merge**

**F**inora, tutto ciò che si è appreso su Oracle, SQL e SQLPLUS riguarda la selezione di dati da tabelle nel database. Questo capitolo mostra in che modo *modificare* i dati in una tabella: come inserire nuove righe, come aggiornare i valori delle colonne nelle righe e come cancellare completamente delle righe. Anche se questi argomenti non sono stati trattati esplicitamente, quasi tutto il materiale appreso in relazione a SQL, compresi tipi di dati, calcoli, formattazione di stringhe, clausole where e simili, può essere utilizzato in questa sede, quindi non rimane molto altro da imparare. Oracle mette a disposizione una funzionalità di database distribuita e trasparente per inserire, aggiornare e cancellare dati in database remoti (trattata nel Capitolo 22). Come si vedrà in questo capitolo, Oracle9i introduce due nuove funzionalità in questo settore: gli inserimenti in più tabelle e il comando merge per supportare le operazioni di inserimento e aggiornamento in un solo comando.

### 15.1 **insert**

Il comando SQL **insert** consente di inserire una riga di informazioni direttamente in una tabella (o indirettamente attraverso una vista). La tabella COMFORT registra le temperature a mezzogiorno e a mezzanotte e le precipitazioni giornaliere, città per città, in quattro date campione nel corso dell'anno:

```
describe COMFORT
```

Nome	Null?	Type
CITTA	NOT NULL	VARCHAR2(13)
DATACAMOIONE	NOT NULL	DATE
MEZZOGIORNO		NUMBER(3,1)
MEZZANOTTE		NUMBER(3,1)
PRECIPITAZIONE		NUMBER

Per aggiungere una nuova riga a questa tabella, si usi l'istruzione seguente:

```
insert into COMFORT
values ('WALPOLE', '21-MAR-01'),
       13.8, 6.6, 0);
```

1 row created.

La parola `values` deve precedere l'elenco di dati da inserire. Una stringa di caratteri dev'essere racchiusa tra apici, mentre i numeri possono stare da soli. I campi devono essere separati da virgolette e devono avere lo stesso ordine delle colonne nella descrizione della tabella.

Una data dev'essere racchiusa tra apici ed espressa nel formato di data predefinito di Oracle. Per inserire una data in un formato diverso occorre utilizzare la funzione `TO_DATE` con una maschera di formattazione, come mostrato di seguito:

```
insert into COMFORT
values ('WALPOLE', TO_DATE('06/22/2001', 'MM/DD/YYYY'),
       13.8, 6.6, 0);
```

1 row created.

## Inserimento di un'ora

Se si inseriscono le date senza specificare i valori dell'ora, viene prodotta un'ora predefinita corrispondente alla mezzanotte, l'inizio del giorno. Se si desidera inserire una data con un'ora differente, è sufficiente ricorrere alla funzione `TO_DATE` e includere un'ora:

```
insert into COMFORT
values ('WALPOLE', TO_DATE('06/22/2001 1:35',
                           'MM/DD/YYYY HH24:MI'), 13.8, 6.6, 0);
```

1 row created.

**NOTA** *Per memorizzare i secondi frazionari, si possono usare i tipi di dati `TIMESTAMP` e `TIMESTAMP WITH TIME ZONE`, come descritto nel Capitolo 9.*

Le colonne possono essere inserite anche in ordine diverso rispetto alla descrizione, se prima (davanti alla parola `values`) si elenca l'ordine dei dati. Questo non modifica l'ordine fondamentale delle colonne nella tabella. Consente semplicemente di elencare i campi dei dati in un ordine diverso (per informazioni sull'inserimento di dati in tipi di dati definiti dall'utente, si rimanda alla Parte quarta).

Si può “inserire” anche un valore `NULL`. In altre parole, la parte sinistra della colonna rimarrà vuota per questa riga, come mostrato di seguito:

```
insert into COMFORT
  (DataCampione, Precipitazione, Citta, Mezzogiorno, Mezzanotte)
values ('23-SEP-01', NULL,
       'WALPOLE', 30.2, 22.3);
```

1 row created.

## insert con select

È anche possibile inserire informazioni che sono state selezionate da una tabella. In questo esempio, vengono inserite alcune colonne selezionate dalla tabella COMFORT, insieme a valori letterali per DataCampione (22-DEC-01) e per Citta (WALPOLE). Si noti la clausola where nell'istruzione select che recupera solo una riga. Se con select fossero state recuperate cinque righe, sarebbero state inserite cinque nuove righe; se ne fossero state ottenute dieci, ne sarebbero state inserite dieci e così via.

```
insert into COMFORT
(DataCampione, Precipitazione, Citta, Mezzogiorno, Mezzanotte)
select '22-DEC-01', Precipitazione,
'WALPOLE', Mezzogiorno, Mezzanotte
from COMFORT
where Citta = 'KEENE' and DataCampione = '22-DEC-01';
```

1 row created.

**NOTA** Non è possibile utilizzare la sintassi insert into...select from con i tipi di dati LONG a meno che non si usi la funzione TO\_LOB per inserire i dati LONG in una colonna LOB.

Naturalmente, non è necessario inserire il valore in una colonna selezionata. È possibile modificare la colonna impiegando nell'istruzione select qualsiasi funzione di stringa, di data o numerica appropriata. In questo caso, vengono inseriti i risultati di queste funzioni. Si può provare a inserire in una colonna un valore che supera la larghezza (per i tipi di dati carattere) o la precisione (per i tipi di dati numerici) della colonna stessa. Tuttavia, ci si deve adattare ai limiti definiti per le colonne. Questi tentativi producono un messaggio di errore “value too large for column” (valore troppo grande per la colonna) o “mismatched datatype” (tipo di dati errato). Se ora si effettua una query sulla tabella COMFORT per la città Walpole, si ottengono i record inseriti:

```
select * from COMFORT
where Citta = 'WALPOLE';
```

CITTA	DATACAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
WALPOLE	21-MAR-01	13.7	6.6	0
WALPOLE	22-JUN-01	13.7	6.6	0
WALPOLE	22-JUN-01	13.7	6.6	0
WALPOLE	23-SEP-01	30.2	22.3	
WALPOLE	22-DEC-01	-21.8	-18.4	3.9

5 rows selected.

Per “22-JUN-01” vengono mostrati due record perché uno di essi ha un componente dell'ora. Per visualizzare l'ora, si usa la funzione TO\_CHAR:

```
select Citta, DataCampione,
TO_CHAR(DataCampione,'HH24:MI:SS') OraGiorno, Mezzogiorno
from COMFORT
where Citta = 'WALPOLE';
```

CITTA	DATA	CAMPAGNA	ORARIO	MEZZOGIORNO
WALPOLE	21-MAR-01	00:00:00		13.7
WALPOLE	22-JUN-01	00:00:00		13.7
WALPOLE	22-JUN-01	01:35:00		13.7
WALPOLE	23-SEP-01	00:00:00		30.2
WALPOLE	22-DEC-01	00:00:00		-21.8

## Uso del suggerimento APPEND per migliorare le prestazioni degli inserimenti

Come si imparerà nel Capitolo 38, Oracle utilizza un ottimizzatore per stabilire il modo più efficiente di eseguire ciascun comando SQL. Per le istruzioni `insert`, Oracle cerca di inserire ogni record nuovo in un blocco di dati esistente che sia stato già allocato nella tabella. Questo piano di esecuzione ottimizza l'uso dello spazio necessario per memorizzare i dati. Tuttavia, potrebbe non offrire prestazioni adeguate per un inserimento con un comando `select` che inserisce più righe. Esiste la possibilità di ignorare il piano di esecuzione con il suggerimento `APPEND`, che consente di migliorare le prestazioni di inserimenti lunghi. Il suggerimento `APPEND` comunica al database di trovare l'ultimo blocco nel quale sono stati inseriti i dati della tabella. I nuovi record vengono inseriti a partire dal blocco successivo, subito dopo quello appena utilizzato. Dato che i dati vengono scritti in nuovi blocchi della tabella, durante l'inserimento il lavoro di gestione dello spazio per il database è minore. Perciò, quando si utilizza `APPEND`, l'operazione può essere portata a termine più velocemente.

Occorre specificare `APPEND` nel comando `insert`. Un suggerimento assomiglia a un commento, inizia con `/*` e termina con `*/`. L'unica differenza è che il set iniziale di caratteri include un segno '+' prima del nome del suggerimento. L'esempio seguente mostra un comando `insert` i cui dati vengono aggiunti alla tabella:

```
insert /*+ APPEND */ into BIBLIOTECA (Titolo)
select Titolo from LIBRO_ORDINE
where Titolo not in (select Titolo from BIBLIOTECA);
```

I record presi dalla tabella `LIBRO_ORDINE` vengono inseriti nella tabella `BIBLIOTECA`. Invece di tentare di riutilizzare lo spazio già usato nella tabella `BIBLIOTECA`, i nuovi record vengono posti alla fine dello spazio di memorizzazione fisico della tabella.

Dal momento che i nuovi record non tentano di riutilizzare lo spazio disponibile che la tabella ha già usato, le esigenze di spazio della tabella `BIBLIOTECA` potrebbero aumentare. In linea di massima, si dovrebbe utilizzare `APPEND` solo quando si inseriscono grandi quantità di dati in tabelle con poco spazio riutilizzabile. Il punto in cui vengono inseriti i record aggiunti prende il nome di *highwatermark* della tabella, e l'unico modo di ripristinare un *highwatermark* consiste nel troncare con `truncate` la tabella stessa. Dato che `truncate` cancella tutti i record e non può essere annullato, è bene assicurarsi che sia disponibile una copia di backup dei dati della tabella prima di utilizzare questo comando. Per ulteriori dettagli, si può cercare la voce `truncate` nel Capitolo 42.

## 15.2 rollback, commit e autocommit

Quando si inseriscono, aggiornano o cancellano dati dal database, si può "invertire" o "annullare" l'operazione con il `rollback`. Questa possibilità può essere molto importante quando si scopre un

errore. Il processo di conferma o rollback dell'operazione è controllato da due comandi SQLPLUS, commit e rollback. Inoltre, SQLPLUS ha la capacità di confermare automaticamente l'operazione senza comunicarlo esplicitamente. Ciò è reso possibile grazie alla caratteristica autocommit di set. Come per altre caratteristiche di set, è possibile visualizzare l'impostazione di autocommit in questo modo:

```
show autocommit
```

```
autocommit OFF
```

OFF è il valore predefinito. Si può anche specificare un numero per il valore autocommit, in modo da determinare il numero di comandi dopo i quali Oracle esegue un commit. Questo significa che gli inserimenti, gli aggiornamenti e le cancellazioni non vengono resi definitivi finché non vengono confermati con commit:

```
commit;
```

```
commit complete
```

Finché non viene eseguito commit, gli altri utenti non possono vedere il lavoro che si sta svolgendo sulle tabelle. Chiunque acceda a queste tabelle continua a ottenere le vecchie informazioni. Le *nuove* informazioni sono visibili ogni volta che si effettua una selezione nella tabella. In effetti, si opera su un'area "provvatoria", con la quale si può interagire finché non si utilizza commit. È possibile effettuare una grande quantità di inserimenti, aggiornamenti e cancellazioni e poi annullare il lavoro (riportando le tabelle alla situazione in cui erano all'inizio) con questo comando:

```
rollback;
```

```
rollback complete
```

Tuttavia, il messaggio "rollback complete" (annullamento completo) può essere fuorviante. In realtà, significa soltanto che Oracle ha annullato ogni operazione che non è stata confermata. Se si conferma una serie di transazioni esplicitamente con la parola commit o implicitamente con un'altra azione, il messaggio "rollback complete" in effetti non significa nulla.

## **Uso di savepoint**

Per annullare solo una parte della serie corrente di transazioni, si possono utilizzare dei *savepoint*. Si considerino i comandi seguenti:

```
insert into COMFORT
values ('WALPOLE', '22-APR-01', 10.5, 1.2, 0);
```

```
savepoint A;
```

```
insert into COMFORT
values ('WALPOLE', '27-MAY-01', 16.8, 2.5, 0);
```

```
savepoint B;
```

```
insert into COMFORT
```

```
values ('WALPOLE', '07-AUG-01', 27.4, 6.6, 0);
```

Ora, si selezionino i dati dalla tabella COMFORT per Walpole:

```
select * from COMFORT
where Citta = 'WALPOLE';
```

CITTA	DATA	CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
WALPOLE	21-MAR-01	13.7	6.6	0	
WALPOLE	22-JUN-01	13.7	6.6	0	
WALPOLE	22-JUN-01	13.7	6.6	0	
WALPOLE	23-SEP-01	30.2	22.3		
WALPOLE	22-DEC-01	-21.7	-18.4	3.9	
WALPOLE	22-APR-01	10.0	-4.0	0	
WALPOLE	27-MAY-01	17.6	1.0	0	
WALPOLE	27-MAY-01	28.3	6.5	0	

L'output mostra i cinque vecchi record più le tre nuove righe che sono state aggiunte. Ora, si annulli solamente l'ultimo inserimento:

```
rollback to savepoint B;
```

Se ora si prova ad effettuare la query sulla tabella COMFORT, si noterà che l'ultimo inserimento è stato annullato, mentre gli altri sono ancora lì:

```
select * from COMFORT
where Citta = 'WALPOLE';
```

CITTA	DATA	CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
WALPOLE	21-MAR-01	13.7	6.6	0	
WALPOLE	22-JUN-01	13.7	6.6	0	
WALPOLE	22-JUN-01	13.7	6.6	0	
WALPOLE	23-SEP-01	30.2	22.3		
WALPOLE	22-DEC-01	-21.7	-18.4	3.9	
WALPOLE	22-APR-01	10.0	-4.0	0	
WALPOLE	27-MAY-01	17.6	1.0	0	

Gli ultimi due record non sono stati ancora confermati; per indurre una conferma di questi record è necessario eseguire un comando commit oppure un altro comando. È ancora possibile annullare il secondo inserimento (to savepoint A) oppure tutti gli inserimenti (tramite un comando rollback).

## commit implicito

Le azioni che fanno in modo che si verifichi un commit, anche senza alcuna istruzione esplicita, sono quit, exit (l'equivalente di quit), e qualsiasi comando del linguaggio DDL (Data Definition Language). Utilizzare uno di questi comandi significa eseguire automaticamente un commit.

## rollback automatico

Se è già stata completata una serie di inserimenti, aggiornamenti o cancellazioni che però non è stata ancora confermata esplicitamente o implicitamente e si incorre in serie difficoltà, come un problema al computer, Oracle annulla automaticamente qualsiasi operazione non confermata. Se il computer o il database si bloccano, questo lavoro di pulizia viene effettuato la prossima volta che il database viene richiamato.

### 15.3 Inserimenti in più tabelle

A partire da Oracle9i è possibile eseguire più inserimenti con un unico comando. Tutti gli inserimenti potranno essere effettuati senza condizioni oppure specificandone alcune, per esempio, l'uso di una clausola `when` per comunicare a Oracle in che modo gestire questi inserimenti multipli. Se si specifica `all`, verranno valutate tutte le clausole `when`; se si specifica `first`, Oracle salta le clausole `when` che seguono quella vera per la riga che viene valutata. Si può utilizzare anche la clausola `else` per indicare a Oracle cosa fare nel caso in cui nessuna clausola `when` venga valutata come vera.

Per illustrare questa funzionalità, si provi a creare una nuova tabella con una struttura leggermente diversa da quella di COMFORT:

```
drop table COMFORT_TEST;
create table COMFORT_TEST (
Citta      VARCHAR2(13) NOT NULL,
DataCampione  DATE NOT NULL,
Misura      VARCHAR2(10),
Valore      NUMBER(3,1)
);
```

`COMFORT_TEST` avrà più record per ogni record presente in `COMFORT`: la colonna `Misura` avrà dei valori come "Mezzanotte", "Mezzogiorno" e "Precipitazione" che consentono di memorizzare un maggior numero di misurazioni per ciascuna città in ogni data campione.

Ora, si popoli la tabella `COMFORT_TEST` con dati presi da `COMFORT`, senza specificare condizioni:

```
insert ALL
    into COMFORT_TEST (Citta, DataCampione, Misura, Valore)
    values (Citta, DataCampione, 'MEZZOGIORNO', Mezzogiorno)
    into COMFORT_TEST (Citta, DataCampione, Misura, Valore)
    values (Citta, DataCampione, 'MEZZANOTTE', Mezzanotte)
    into COMFORT_TEST (Citta, DataCampione, Misura, Valore)
    values (Citta, DataCampione, 'PRECIPITAZIONE', Precipitazione)
select Citta, DataCampione, Mezzogiorno, Mezzanotte, Precipitazione
  from COMFORT
 where Citta = 'KEENE';
```

12 rows created.

Questa query comunica a Oracle di inserire più righe per ogni riga restituita dalla tabella `COMFORT`. La query su `COMFORT` restituisce quattro righe. La prima clausola `into` inserisce il valore di Mezzogiorno, insieme a una stringa di testo, "MEZZOGIORNO", nella colonna `Misura`.

ra. La seconda clausola into inserisce i valori di Mezzanotte, mentre la terza inserisce i valori per Precipitazione, come mostrato nella query su COMFORT\_TEST che segue l'inserimento:

```
select * from COMFORT_TEST;
```

CITTA	DATA CAMPIONE	MISURA	VALORE
KEENE	21-MAR-01	MEZZOGIORNO	4.4
KEENE	22-JUN-01	MEZZOGIORNO	29.5
KEENE	23-SEP-01	MEZZOGIORNO	37.6
KEENE	22-DEC-01	MEZZOGIORNO	-21.7
KEENE	21-MAR-01	MEZZANOTTE	-18.4
KEENE	22-JUN-01	MEZZANOTTE	19.3
KEENE	23-SEP-01	MEZZANOTTE	28.1
KEENE	22-DEC-01	MEZZANOTTE	-18.4
KEENE	21-MAR-01	PRECIPITAZIONE	4.4
KEENE	22-JUN-01	PRECIPITAZIONE	1.3
KEENE	23-SEP-01	PRECIPITAZIONE	
KEENE	22-DEC-01	PRECIPITAZIONE	3.9

12 rows selected.

Cosa sarebbe accaduto se al posto della parola chiave `all` fosse stata utilizzata la parola chiave `first`? A meno che non si utilizzino delle clausole `when`, non è possibile usare la parola chiave `first`. L'esempio seguente mostra l'uso della clausola `when` per limitare il numero di inserimenti eseguiti nel comando di inserimenti multipli. Per questo esempio, viene utilizzata la parola chiave `all`:

```
delete from COMFORT_TEST;
commit;

insert ALL
    when Mezzogiorno > 27 then
        into COMFORT_TEST (Città, DataCampione, Misura, Valore)
        values (Città, DataCampione, 'MEZZOGIORNO', Mezzogiorno)
    when Mezzanotte > 21 then
        into COMFORT_TEST (Città, DataCampione, Misura, Valore)
        values (Città, DataCampione, 'MEZZANOTTE', Mezzanotte)
    when Precipitazione is not null then
        into COMFORT_TEST (Città, DataCampione, Misura, Valore)
        values (Città, DataCampione, 'PRECIPITAZIONE', Precipitazione)
select Città, DataCampione, Mezzogiorno, Mezzanotte, Precipitazione
    from COMFORT
   where Città = 'KEENE';
```

6 rows created.

Quali sono le sei righe inserite? I due valori di Mezzogiorno che soddisfano questa condizione:

```
when Mezzogiorno > 27 then
```

il valore di Mezzanotte che soddisfa questa condizione:

```
when Mezzanotte > 21 then
```

infine, i tre valori di Precipitazione che soddisfano questa condizione:

when Precipitazione is not null then

I risultati sono visibili in COMFORT\_TEST:

```
select * from COMFORT_TESTT;
```

CITTA	DATA CAMPIONE	MISURA	VALORE
KEENE	22-JUN-01	MEZZOGIORNO	29.5
KEENE	23-SEP-01	MEZZOGIORNO	37.6
KEENE	23-SEP-01	MEZZANOTTE	28.1
KEENE	21-MAR-01	PRECIPITAZIONE	4.4
KEENE	22-JUN-01	PRECIPITAZIONE	1.3
KEENE	22-DEC-01	PRECIPITAZIONE	3.9

Cosa sarebbe accaduto con la parola chiave first?

```
delete from COMFORT_TEST;
commit;
```

```
insert FIRST
when Mezzogiorno > 27 then
    into COMFORT_TEST (Città, DataCampione, Misura, Valore)
    values (Città, DataCampione, 'MEZZOGIORNO', Mezzogiorno)
when Mezzanotte > 21 then
    into COMFORT_TEST (Città, DataCampione, Misura, Valore)
    values (Città, DataCampione, 'MEZZANOTTE', Mezzanotte)
when Precipitazione is not null then
    into COMFORT_TEST (Città, DataCampione, Misura, Valore)
    values (Città, DataCampione, 'PRECIPITAZIONE', Precipitazione)
select Città, DataCampione, Mezzogiorno, Mezzanotte, Precipitazione
from COMFORT
where Città = 'KEENE';
```

4 rows created.

Sono state inserite quattro righe:

```
select * from COMFORT_TEST;
```

CITTA	DATA CAMPIONE	MISURA	VALORE
KEENE	21-MAR-01	PRECIPITAZIONE	4.4
KEENE	22-DEC-01	PRECIPITAZIONE	3.9
KEENE	22-JUN-01	MEZZOGIORNO	29.5
KEENE	23-SEP-01	MEZZOGIORNO	37.6

Cosa è accaduto al valore di MEZZANOTTE? L'unico record che ha passato la clausola when di MEZZANOTTE:

when Mezzanotte > 21 then

ha passato anche la clausola when di MEZZOGIORNO:

when Mezzogiorno > 27 then

quindi, è stato inserito il suo valore di Mezzogiorno (37.6). Dato che è stata impiegata la parola chiave `first`, e che la condizione valutata per prima (Mezzogiorno) era vera, Oracle non ha controllato le altre condizioni relative a questa riga. Lo stesso processo si è verificato con le misure di PRECIPITAZIONE, l'altro valore non NULL di Precipitazione si trovava sullo stesso record di Mezzogiorno, pari a 29.5.

Cosa sarebbe accaduto se nessuna delle condizioni fosse stata soddisfatta? Per visualizzare questa opzione si crei una terza tabella, COMFORT2, con la medesima struttura di COMFORT:

```
create table COMFORT2 (
    Citta          VARCHAR2(13) NOT NULL,
    DataCampione   DATE NOT NULL,
    Mezzogiorno    NUMBER(3,1),
    Mezzanotte     NUMBER(3,1),
    Precipitazione NUMBER
);
```

Ora, verrà eseguito un inserimento per tutte le città (con la clausola `first`) e verrà introdotta una clausola `else` secondo cui qualsiasi riga non soddisfi le condizioni verrà inserita nella tabella COMFORT2. Per questo esempio, le condizioni `when` sono state modificate per limitare il numero di righe che soddisfano le condizioni.

```
delete from COMFORT_TEST;
delete from COMFORT2;
commit;

insert FIRST
    when Mezzogiorno > 27 then
        into COMFORT_TEST (Citta, DataCampione, Misura, Valore)
        values (Citta, DataCampione, 'MEZZOGIORNO', Mezzogiorno)
    when Mezzanotte > 35 then
        into COMFORT_TEST (Citta, DataCampione, Misura, Valore)
        values (Citta, DataCampione, 'MEZZANOTTE', Mezzanotte)
    when Precipitazione > 100 then
        into COMFORT_TEST (Citta, DataCampione, Misura, Valore)
        values (Citta, DataCampione, 'PRECIPITAZIONE', Precipitazione)
    else
        into COMFORT2
select Citta, DataCampione, Mezzogiorno, Mezzanotte, Precipitazione
  from COMFORT
 where Citta = 'KEENE';

4 rows created.
```

Il risultato indica quante righe sono state create, ma non in quale tabella sono state inserite. Il totale indicato riguarda tutti gli inserimenti. In questo caso, due record sono stati inseriti in COMFORT\_TEST e altri due in COMFORT2 a causa della presenza della condizione `else`:

```
select * from COMFORT_TEST;
```

CITTA	DATA CAMPIONE	MISURA	VALORE
KEENE	22-JUN-01	MEZZOGIORNO	29.5
KEENE	23-SEP-01	MEZZOGIORNO	37.6

```
select * from COMFORT2;
```

CITTA	DATA	CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
KEENE	21-MAR-01	4.4	-18.4	4.4	
KEENE	22-DEC-01	-21.7	-18.4	3.9	

## 15.4 delete

La rimozione di una o più righe da una tabella richiede il comando **delete**, come mostrato negli esempi dell'ultimo paragrafo. La clausola **where** è essenziale per eliminare solo le righe desiderate. **delete** senza una clausola **where** svuota completamente la tabella. Il comando seguente elimina le voci relative a Walpole dalla tabella COMFORT.

```
delete from COMFORT where Citta = 'WALPOLE';
```

Ovviamente, una clausola **where** in **delete**, proprio come in **update** o **select** che faccia parte di **insert**, può contenere tutta la logica di una qualsiasi istruzione **select** e può includere subquery, unioni, intersezioni e così via. Si può sempre annullare un comando **insert**, **update** o **delete** errato ma, in realtà, prima di modificare il database, conviene sempre provare con **select** per assicurarsi di fare la cosa giusta.

Ora che sono state cancellate le righe dove **Citta='WALPOLE'**, si può verificare l'effetto di **delete** con una semplice query:

```
select * from COMFORT
where Citta = 'WALPOLE';
```

no rows selected

Ora, si provi ad annullare **delete** ed eseguire di nuovo la stessa query:

```
rollback;
rollback complete
```

```
select * from COMFORT
where Citta = 'WALPOLE';
```

CITTA	DATA	CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
WALPOLE	21-MAR-01	13.7	6.6	0	
WALPOLE	22-JUN-01	13.7	6.6	0	
WALPOLE	22-JUN-01	13.7	6.6	0	
WALPOLE	23-SEP-01	30.2	22.3		
WALPOLE	22-DEC-01	-21.8	-18.4	3.9	
WALPOLE	22-APR-01	10.0	-4.0	0	
WALPOLE	27-MAY-01	17.6	1.0	0	

7 rows selected.

Questo dimostra che il recupero è possibile, purché non si sia verificato un **commit**.

Un altro comando utilizzato per cancellare record, `truncate`, non si comporta come `delete`. Mentre `delete` consente di confermare o annullare la cancellazione, `truncate` elimina automaticamente tutti i record dalla tabella. L'azione del comando `truncate` non può essere annullata o confermata; i record troncati sono irrecuperabili. Per ulteriori dettagli, consultare la voce `truncate` nel Capitolo 42.

## 15.5 **update**

`update` richiede di impostare valori specifici per ogni colonna che si desidera modificare e di specificare su quale riga o su quali righe si intende operare utilizzando una clausola `where` costruita attentamente:

```
update COMFORT set Precipitazione = .5, Mezzanotte = 73.1  
where Citta = 'KEENE'  
and DataCampione = '22-DEC-2001';
```

1 row updated.

Ecco l'effetto, mostrato nel record 22-DEC-01:

```
select * from COMFORT  
where Citta = 'KEENE';
```

CITTA	DATA CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
KEENE	21-MAR-01	4.4	-18.4	4.4
KEENE	22-JUN-01	29.5	19.2	1.3
KEENE	23-SEP-01	37.6	28.1	
KEENE	22-DEC-01	-21.7	22.8	.5

4 rows selected.

Cosa si può fare se si scopre che il termometro utilizzato a Keene riporta le temperature costantemente aumentate di un grado? Si possono anche svolgere calcoli, funzioni di stringa e qualsiasi altra funzione legittima nell'impostazione di un valore per `update` (proprio come per `insert` o per la clausola `where` in `delete`). Nell'esempio seguente, tutte le temperature di Keene sono state ridotte di un grado:

```
update COMFORT set Mezzanotte = Mezzanotte -1, Mezzogiorno = Mezzogiorno -1  
where Citta = 'KEENE';
```

4 rows updated.

Ecco l'effetto dell'aggiornamento:

```
select * from COMFORT  
where Citta = 'KEENE';
```

CITTA	DATA CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
KEENE	21-MAR-01	3.4	-17.4	4.4
KEENE	22-JUN-01	28.5	18.2	1.3

```

KEENE      23-SEP-01      36.6      27.1
KEENE      22-DEC-01     -20.7      21.8      .5

```

4 rows selected.

**NOTA** Se l'aggiornamento non rispetta le definizioni o i limiti di colonna, non andrà a buon fine. In questo caso, l'impostazione di Mezzogiorno o Mezzanotte a valori come 100 o maggiori viola la scala numerica delle colonne.

Anche qui, come in delete, la clausola where è fondamentale. Senza di essa, viene aggiornata ogni riga del database. Con una clausola where costruita in modo non corretto, verranno aggiornate le righe errate, spesso in modo difficile da scoprire o risolvere, soprattutto se l'operazione è stata confermata con un commit. Quando si effettuano degli aggiornamenti, conviene sempre utilizzare l'impostazione set feedback on ed esaminare il risultato per assicurarsi che il numero di righe aggiornate sia quello previsto. È poi utile effettuare una query sulle righe dopo l'aggiornamento per controllare se si sono verificati i cambiamenti previsti.

## update con un'istruzione select incorporata

È possibile impostare i valori in un comando update incorporandovi un'istruzione select proprio nel mezzo. Si osservi che questa select ha la propria clausola where, che preleva la temperatura per la città di MANCHESTER dalla tabella CLIMA, e che anche update ha la propria clausola where che influisce solo sulla città KEENE in un certo giorno:

```

update COMFORT set Mezzanotte =
  (select Temperatura
   from CLIMA
   where Citta = 'MANCHESTER')
where Citta = 'KEENE'
  AND DataCampione = '22-DEC-01';

```

1 row updated.

Ecco l'effetto dell'aggiornamento:

```

select * from COMFORT
where Citta = 'KEENE';

```

CITTA	DATA CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
KEENE	21-MAR-01	3.4	-19.0	4.4
KEENE	22-JUN-01	28.5	18.7	1.3
KEENE	23-SEP-01	36.6	81.6	
KEENE	22-DEC-01	-20.7	27.5	.5

4 rows selected.

Quando si utilizzano delle subquery con comandi update, occorre assicurarsi che la subquery non restituisca più di un record per ognuno dei record aggiornati, altrimenti l'aggiornamento fallisce. Per maggiori dettagli sulle query correlate, si rimanda al Capitolo 12.

Inoltre, è possibile utilizzare un'istruzione select incorporata per aggiornare più colonne contemporaneamente. Le colonne devono essere racchiuse fra parentesi e separate da una virgola, come mostrato di seguito:

```
update COMFORT set (Mezzogiorno, Mezzanotte) =
  (select Umidita, Temperatura
   from CLIMA
   where Citta = 'MANCHESTER')
where Citta = 'KEENE'
  AND DataCampione = '22-DEC-01';
```

1 row updated.

Ed ecco il risultato:

```
select * from COMFORT
where Citta = 'KEENE';
```

CITTA	DATA CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
KEENE	21-MAR-01	3.4	-19.0	4.4
KEENE	22-JUN-01	28.5	18.7	1.3
KEENE	23-SEP-01	36.6	81.6	
KEENE	22-DEC-01	-20.7	27.5	.5

4 rows selected.

## **update con NULL**

È anche possibile aggiornare una tabella e impostare una colonna uguale a NULL. Questo è il solo caso in cui si può utilizzare con NULL il segno di uguale, invece della parola “is”. Per esempio, in:

```
update COMFORT set Mezzogiorno = NULL
where Citta = 'KEENE'
  AND DataCampione = '22-DEC-01';
```

1 row updated.

si imposta a NULL la temperatura di mezzogiorno per Keene del 22.12.01.

**NOTA** *I problemi principali con insert, update e delete consistono nell'attenta costruzione delle clausole where in modo che operino (o inseriscano) solo sulle righe veramente desiderate, e nel normale utilizzo di funzioni SQL in queste istruzioni. È estremamente importante fare attenzione a non confermare l'operazione prima di essere sicuri che sia corretta. Questi comandi estendono la potenza di Oracle ben al di là delle semplici query e consentono di gestire direttamente la manipolazione dei dati.*

## **15.6 Uso del comando merge**

A partire da Oracle9i è possibile utilizzare il comando merge per eseguire inserimenti e aggiornamenti in un'unica tabella e con un solo comando. In base alle condizioni specificate, Oracle prenderà i dati di origine, una tabella, una vista oppure una query, e aggiornerà i valori già esistenti se le condizioni imposte sono soddisfatte. In caso contrario, la riga verrà inserita.

Per questo esempio, si provi a modificare le righe nella tabella COMFORT2 create in precedenza:

```
update COMFORT2 set Mezzogiorno = 12.7;
insert into COMFORT2 values ('KEENE', '16-MAY-01', 12.7, 12.7, 1);
commit;
```

Ora, i dati di COMFORT2 dovrebbero avere l'aspetto seguente:

```
select * from COMFORT2;
```

CITTA	DATACAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
KEENE	21-MAR-01	12.7	-19	4.4
KEENE	22-DEC-01	12.7	18.9	.5
KEENE	16-MAY-01	12.7	12.7	1

I dati relativi a Keene in COMFORT sono:

```
select * from COMFORT where Citta = "KEENE";
```

CITTA	DATACAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
KEENE	21-MAR-01	3.4	-19	4.4
KEENE	22-JUN-01	28.5	18.7	1.3
KEENE	23-SEP-01	36.6	27.5	
KEENE	22-DEC-01		18.9	.5

Con COMFORT2 utilizzata come origine di dati, è possibile eseguire un'unione con il comando merge sulla tabella COMFORT. Per le righe che corrispondono alle voci '21-MAR-01' e '22-DEC-01' verranno aggiornati i valori di Mezzogiorno di COMFORT. Le righe che esistono solamente in COMFORT2 (per esempio, '16-MAY-01') verranno inserite in COMFORT. Il listato seguente mostra il comando da usare.

```
merge into COMFORT  C1
using (select Citta, DataCampione, Mezzogiorno, Mezzanotte,
          Precipitazione from COMFORT2)  C2
  on (C1.Citta = C2.Citta and C1. DataCampione=C2. DataCampione)
when matched then
    update set Mezzogiorno = C2.Mezzogiorno
when not matched then
    insert (C1.Citta, C1. DataCampione, C1.Mezzogiorno,
            C1.Mezzanotte, C1.Precipitazione)
    values (C2.Citta, C2. DataCampione, C2.Mezzogiorno,
            C2.Mezzanotte, C2.Precipitazione);
```

3 rows merged.

L'output indica il numero di righe elaborate dalla tabella sorgente, ma non dice nulla riguardo il numero delle righe inserite o aggiornate. È possibile visualizzare le modifiche eseguendo una query su COMFORT (si notino i record di Mezzogiorno=55):

```
select * from COMFORT where Citta = 'KEENE';
```

CITTA	DATA	CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
KEENE	21-MAR-01	12.7	-19	4.4	
KEENE	22-JUN-01	28.5	18.7	1.3	
KEENE	23-SEP-01	36.6	27.5		
KEENE	22-DEC-01	12.7	18.9	.5	
KEENE	16-MAY-01	12.7	55		1

Si analizzi il comando per capire come si è arrivati a questo risultato. Nella prima linea, viene nominata la tabella di destinazione e le viene assegnato un alias, C1:

```
merge into COMFORT    C1
```

Nelle due linee successive, la clausola using fornisce i dati di origine per l'unione, e alla sorgente viene assegnato l'alias C2:

```
using (select Citta, DataCampione, Mezzogiorno, Mezzanotte,
       Precipitazione from COMFORT2)  C2
```

La condizione per l'unione viene quindi specificata nella clausola on. Se i valori Citta e DataCampione dei dati di origine corrispondono a quelli nella tabella di destinazione, i dati verranno aggiornati. La clausola when matched then, seguita da un comando update, indica a Oracle quali colonne aggiornare nella tabella sorgente:

```
on (C1.Citta = C2.Citta and C1. DataCampione=C2. DataCampione)
when matched then
    update set Mezzogiorno = C2.Mezzogiorno
```

Se non esiste alcuna corrispondenza, allora si dovrebbe inserire la riga, come specificato nella clausola when not matched:

```
when not matched then
    insert (C1.Citta, C1.DataCampione, C1.Mezzogiorno,
           C1.Mezzanotte, C1.Precipitazione)
    values (C2.Citta, C2.DataCampione, C2.Mezzogiorno,
            C2.Mezzanotte, C2.Precipitazione);
```

È possibile usare il comando merge per semplificare le operazioni in cui vengono inserite e aggiornate molte righe tratte da un'unica sorgente. Come avviene per tutti i comandi update, si dovrà prestare molta attenzione a utilizzare le clausole where appropriate nella clausola using dei comandi merge.

## Capitolo 16

# Uso avanzato di funzioni e variabili

- 16.1 **Funzioni in order by**
- 16.2 **Diagrammi a barre e grafici**
- 16.3 **Uso di TRANSLATE**
- 16.4 **Copia e incollamento complessi**
- 16.5 **Conteggio delle occorrenze di stringhe  
in stringhe più lunghe**
- 16.6 **Ulteriori informazioni sulle variabili**

**N**ei capitoli precedenti sono stati presentati esempi e definizioni per funzioni di caratteri, numeri e date e per l'utilizzo di variabili. È stata fornita un'ampia gamma di esempi, dai più semplici ai più complessi, e una spiegazione sufficiente a consentire di costruire combinazioni di funzioni anche abbastanza complesse.

Questo capitolo amplia alcuni degli argomenti esposti in precedenza, mostrando esempi di come le funzioni possano essere combinate per risolvere problemi più difficili. Oracle ha reso più semplice la risoluzione di alcuni di questi problemi grazie all'uso di SQL e gli esempi in questo capitolo possono aiutare a sviluppare le capacità per risolvere anche i problemi reali.

## 16.1 Funzioni in order by

Per cambiare l'ordinamento della sequenza, in order by si possono utilizzare anche le funzioni. In questo esempio vengono impiegate le funzioni SUBSTR per mettere l'elenco degli autori in ordine alfabetico in base al nome:

```
select Autore
      from RIVISTA
     order by SUBSTR(Autore,INSTR(Autore,',')+2);
```

AUTORE

```
-----  
WHITEHEAD, ALFRED  
BONHOEFFER, DIETRICH  
CHESTERTON, G.K.  
RUTH, GEORGE HERMAN  
CROOKES, WILLIAM
```

## 16.2 Diagrammi a barre e grafici

In SQLPLUS è anche possibile produrre semplici diagrammi a barre e grafici, utilizzando un calcolo numerico nella funzione LPAD. Il seguente diagramma a barre mostra il numero medio dei giorni di durata del prestito dei libri per ogni persona. Per prima cosa, occorre esaminare i comandi di formattazione delle colonne impiegati:

```
column Nome format a16
column GiorniFuori Format 999
column Grafico Heading 'Giorno|   1   2   3   4   5   6   7-
|....0....0....0....0....0....0' justify c
column Grafico format a35
```

I primi due comandi (linee) sono chiari, mentre il terzo richiede alcune spiegazioni. Il trattino alla fine della terza linea comunica a SQLPLUS che il comando di colonna continua alla linea successiva. Se alla fine di una linea compare un trattino, SQLPLUS presuppone che segua un'altra linea dello stesso comando. Ecco l'istruzione SQL che ha prodotto il diagramma a barre orizzontali del listato perduto:

```
select Nome,
       AVG(DataRestituzione-DataPrestito) GiorniFuori,
       LPAD('o',ROUND(AVG(DataRestituzione-DataPrestito)/2,0),'o')
             Grafico
  from BIBLIOTECA_PRESTITO
 group by Nome
 order by AVG(DataRestituzione-DataPrestito);
```

NOME	GIORNIFUORI	Giorno						
		1	2	3	4	5	6	7
DORAH TALBOT	13	oooooooo						
EMILY TALBOT	14	oooooooo						
JED HOPKINS	18	oooooooooooo						
GERHARDT KENTGEN	19	oooooooooooo						
PAT LAVAY	21	oooooooooooooo						
FRED FULLER	24	oooooooooooooo						
ROLAND BRANDT	52	oooooooooooooooooooo	oooooooooooo	oooooooooooo	oooooooooooo	oooooooooooo	oooooooooooo	

7 rows selected.

In questo esempio, LPAD viene utilizzata più o meno come nel Capitolo 13 in relazione all'albero genealogico delle mucche e dei tori. In sostanza, la colonna qui è costituita da una ‘o’ minuscola e viene riempita a sinistra con un certo numero di ‘o’ aggiuntive, fino alla larghezza massima determinata da ROUND(AVG(DataRestituzione-DataPrestito)/2,0).

Si noti che la scala dell'intestazione di colonna viene incrementata di 2. Una semplice modifica all'istruzione SQL produce un grafico classico invece di un diagramma a barre. La colonna letterale viene impostata con una ‘x’ minuscola invece della ‘o’, e il riempimento a sinistra è effettuato con spazi.

```
select Nome,
       AVG(DataRestituzione-DataPrestito) GiorniFuori,
       LPAD('x',ROUND(AVG(DataRestituzione-DataPrestito)/2,0),' ')
             AS Grafico
```

```
from BIBLIOTECA_PRESTITO
group by Nome
order by AVG(DataRestituzione-DataPrestito);
```

NOME	GIORNIFUORI	Giorno						
		1	2	3	4	5	6	7
DORAH TALBOT	13		x					
EMILY TALBOT	14		x					
JED HOPKINS	18			x				
GERHARDT KENTGEN	19		x					
PAT LAVAY	21			x				
FRED FULLER	24				x			
ROLAND BRANDT	52						x	

7 rows selected.

Un altro modo per rappresentare i dati in un grafico consiste nel basarsi sulla distribuzione invece che sulle persone. Innanzitutto viene creata una vista che colloca ciascun numero di giorni nel proprio decile. Pertanto, 13, 14, 18 e 19 diventano 10; i giorni da 20 a 29 diventano 20; quelli da 30 a 39 diventano 30 e così via:

```
create or replace view IntervalloGiorniFuori as
select ROUND((DataRestituzione-DataPrestito),-1) Decile,
       COUNT(*) Contatore
  from BIBLIOTECA_PRESTITO
 group by ROUND((DataRestituzione-DataPrestito),-1);
```

Poi, viene configurata un'intestazione della colonna, simile a quella precedente ma più corta e con incrementi di 1:

```
column Grafico Heading 'Conteggio'          1    1|....5....0....5'
       justify c
column Grafico format a15
```

La prossima istruzione SQL determina il numero dei valori rappresentati in ogni decile.

```
select Decile, Contatore,
       LPAD('o',Contatore,'o') Grafico
  from INTERVALLOGIORNIFUORI;
```

DECILE	CONTATORE	Conteggio	
		1	1
10	8	ooooooo	0
20	5	oooo	0
30	3	ooo	0
50	2	oo	0
60	1	o	0

Se una delle date fosse NULL, l'output di group by includerebbe un gruppo di valori NULL, con un conteggio e una visualizzazione simili a quelli mostrati in questo listato. Se si desidera personalizzare il modo in cui Oracle gestisce i libri con i valori DataRestituzione NULL, si potrebbe utilizzare la funzione NVL per sostituire il valore NULL con uno a scelta (come SysDate).

Per una logica di sostituzione dei dati più complessa, si possono usare le funzioni DECODE e CASE, come spiegato più avanti in questo capitolo e descritto in dettaglio nel Capitolo 17.

### 16.3 Uso di TRANSLATE

TRANSLATE converte i caratteri di una stringa in caratteri diversi, in base a un piano di sostituzione, da *if* in *then*:

```
TRANSLATE(stringa,if,then)
```

Nell'istruzione SQL seguente, vengono sostituite le lettere in una frase. Ogni volta che viene rilevata una 'T' maiuscola, questa viene sostituita con un'altra 'T' maiuscola. In effetti non cambia nulla. Ogni volta che viene rilevata una vocale maiuscola, questa viene sostituita dalla corrispondente vocale minuscola. Ogni lettera che non si trova nella stringa TAEIOU non viene cambiata. Quando viene trovata una lettera presente in TAEIOU, ne viene controllata la posizione in questa stringa e viene sostituita. Pertanto, la lettera 'E', alla posizione 3 in TAEIOU, viene sostituita con una 'e', alla posizione 3 in Taeiou:

```
select TRANSLATE('ATTACCO ALLE VOCALI','TAEIOU','Taeiou')
  from DUAL;

TRANSLATE('ATTACCOALLEVOCAL
-----
aTTaCCo aLLe VoCaLi
```

### Eliminazione di caratteri

Allargando l'applicazione di questa logica, cosa accade se la stringa *if* è TAEIOU e la stringa *then* è soltanto T? Se si controlla la lettera 'E' (come nella parola "ALLE"), questa si trova nella posizione 3 di TAEIOU. La posizione 3 nella stringa *then* (che è costituita solo dalla lettera *T*) non esiste, quindi alla posizione non vi è nulla. Al posto della 'E' non viene inserito nulla. Lo stesso procedimento viene applicato a tutte le vocali. Esse sono presenti nella stringa *if*, ma non in quella *then*. Quindi le vocali scompaiono, come mostrato di seguito:

```
select TRANSLATE('ATTACCO ALLE VOCALI','TAEIOU','T')
  from DUAL;

TRANSLATE('ATTACCOALLEVOCAL
-----
TTCC LL VCL
```

Questa caratteristica di TRANSLATE, ovvero la capacità di eliminare caratteri da una stringa, può essere molto utile per mettere a punto i dati. Si può riprendere la tabella con i titoli di rivista descritta nel Capitolo 7:

```
select Titolo from RIVISTA;
TITOL
-----
THE BARBERS WHO SHAVE THEMSELVES.
"HUNTING THOREAU IN NEW HAMPSHIRE"
```

---

THE ETHNIC NEIGHBORHOOD  
RELATIONAL DESIGN AND ENTHALPY  
"INTERCONTINENTAL RELATIONS."

Il metodo adottato nel Capitolo 7 per rimuovere i punti e le virgolette era costituito da una combinazione di LTRIM e RTRIM:

```
select LTRIM( RTRIM(Titolo,'.'), '' ) from RIVISTA;
```

Lo stesso risultato può essere realizzato con una singola istruzione TRANSLATE:

```
select TRANSLATE(Titolo,'T','.') TITOLO
      from RIVISTA;
```

TITOLO

---

-----  
THE BARBERS WHO SHAVE THEMSELVES  
HUNTING THOREAU IN NEW HAMPSHIRE  
THE ETHNIC NEIGHBORHOOD  
RELATIONAL DESIGN AND ENTHALPY  
INTERCONTINENTAL RELATIONS

Nel listato si deve includere un carattere nella stringa *then*. In questo caso, la lettera *T* si converte nella lettera *t*. Tutti gli altri caratteri nella stringa *if* vengono eliminati.

## 16.4 Copia e incollamento complessi

La tabella NOME contiene un elenco di nomi, simile a quello che si potrebbe ricevere da un'azienda che commercializza liste di indirizzi o da un'altra applicazione. Il nome, il cognome e le iniziali sono tutti in una colonna:

```
column Nome format a25
```

```
select Nome from NOME;
```

NOME

---

-----  
HORATIO NELSON  
VALDO  
MARIE DE MEDICIS  
FLAVIUS JOSEPHUS  
EDYTHE P. M. GAMMIERE

Si supponga di voler tagliare e incollare questi nomi, per inserirli in una tabella con le colonne Nome e Cognome. Come si fa? La tecnica appresa nel Capitolo 7 richiedeva l'uso di INSTR per individuare uno spazio, nonché l'uso del numero restituito in una funzione SUBSTR per tagliare la parte compresa fino allo spazio. Qui viene presentato un tentativo di fare la stessa cosa per il nome:

```
select SUBSTR(Nome,1,INSTR(Nome,' '))
      from NOME;
```

```
SUBSTR(NOME,1,INSTR(NOME,  
-----  
HORATIO  
  
MARIE  
FLAVIUS  
EDYTHE
```

VALDO è sparito! Il problema è che questi nomi non sono coerenti come quelli degli autori riportati nel Capitolo 7; uno di essi (probabilmente un mago) è costituito da un nome solo, perciò non vi è alcuno spazio che possa essere trovato da INSTR. Quando INSTR non riesce nella ricerca, restituisce uno zero. SUBSTR, applicata al nome VALDO dalla posizione 1 alla posizione 0 non produce nulla, perciò questo nome scompare. Questo problema può essere risolto con DECODE. Invece di:

```
INSTR(Nome,'')
```

occorre inserire l'intera espressione, come questa:

```
DECODE(INSTR(Nome,''),0,99,INSTR(Nome,''))
```

La sintassi di DECODE è la seguente:

```
DECODE(valore,if1,then1[,if2,then2,if3,then3]...,[else])
```

Nell'esempio precedente, DECODE controlla il valore di INSTR(Nome,''). Se questo valore è uguale a 0 DECODE sostituisce 99, altrimenti restituisce il valore predefinito, che coincide con INSTR(Nome,''). La scelta di 99 come valore sostitutivo è arbitraria. In questo modo, viene creata una reale funzione SUBSTR per VALDO, simile a questa:

```
SUBSTR('VALDO',1,99)
```

Questa funziona perché SUBSTR ritaglia il testo, dal numero iniziale, 1, fino al numero finale o alla fine della stringa, quello dei due che compare per primo. Con la funzione DECODE, i nomi vengono restituiti correttamente:

```
select SUBSTR(Nome,1,  
              DECODE(INSTR(Nome,''),0,99,INSTR(Nome,''))))  
  from NOME;  
  
SUBSTR(NOME,1,DECODE(INST  
-----  
HORATIO  
VALDO  
MARIE  
FLAVIUS  
EDYTHE
```

E i cognomi? Si potrebbe utilizzare di nuovo INSTR per ricercare uno spazio, e usare la posizione di questo spazio nella stringa (+1) come punto iniziale per SUBSTR. Per SUBSTR non è necessario alcun punto finale, perché si vuole arrivare fino alla fine del nome. Ecco cosa accade:

```
select SUBSTR(Nome,INSTR(Nome,'')+1)  
  from NOME;
```

```
SUBSTR(NOME,INSTR(NOME,''
```

```
-----  
NELSON  
VALDO  
DE MEDICIS  
JOSEPHUS  
P. M. GAMMIERE
```

Questo non va bene. Una soluzione consiste nell'utilizzare tre funzioni INSTR, cercando in successione la prima, la seconda o la terza occorrenza di uno spazio nel nome. Ognuna di queste funzioni restituisce la posizione dove ha trovato uno spazio, oppure uno zero se non ne ha trovato alcuno. In un nome con un solo spazio, la seconda e la terza funzione INSTR restituiscono entrambe uno zero. Di conseguenza, la funzione GREATEST preleva il numero restituito da INSTR che ha trovato lo spazio più lontano nel nome:

```
select SUBSTR(Nome,  
GREATEST(INSTR(Nome,' '),INSTR(Nome,' ',1,2),INSTR(Nome,' ',1,3)) +1)  
from NOME;
```

```
SUBSTR(NOME,GREATEST(INST
```

```
-----  
NELSON  
VALDO  
MEDICIS  
JOSEPHUS  
GAMMIERE
```

A parte il fatto che si è ottenuto di nuovo VALDO, questo codice funziona (GREATEST avrebbe potuto essere utilizzata in modo simile al posto di DECODE anche nell'esempio precedente). Esiste un secondo metodo, più semplice:

```
select SUBSTR(Nome,INSTR(Nome,' ', -1)+1)  
from NOME;
```

```
SUBSTR(NOME,INSTR(NOME,''
```

```
-----  
NELSON  
VALDO  
MEDICIS  
JOSEPHUS  
GAMMIERE
```

Il numero **-1** in INSTR indica che la ricerca nella colonna Nome dev'essere iniziata dalla posizione finale, spostandosi *all'indietro*, o da destra verso sinistra. Quando trova lo spazio, INSTR restituisce la sua posizione, contando come al solito da sinistra. **-1** induce semplicemente INSTR a iniziare la ricerca dalla fine invece che dall'inizio. Un **-2** farebbe iniziare la ricerca dalla seconda posizione a partire dalla fine, e così via.

Il valore **+1** in SUBSTR ha lo stesso scopo che nell'esempio precedente: una volta trovato lo spazio, SUBSTR deve spostarsi di una posizione verso destra per iniziare a tagliare il nome. Se non viene trovato alcuno spazio, INSTR restituisce 0, quindi SUBSTR inizia con la posizione 1. Questo è il motivo per cui VALDO è nell'elenco.

Come si fa a eliminare VALDO? Occorre aggiungere una posizione finale a SUBSTR, invece di mantenere quella predefinita (che va automaticamente fino alla fine). La posizione finale viene trovata utilizzando il comando seguente:

```
DECODE(INSTR(Nome, ' '), 0, 0, LENGTH(Nome))
```

che comunica, “Trovare la posizione di uno spazio in Nome”. Se la posizione è zero, viene restituito zero; altrimenti viene restituita la lunghezza di Nome. Per VALDO, DECODE produce 0 come posizione finale per SUBSTR, così non viene visualizzato nulla. Per qualsiasi altro nome, poiché vi è sicuramente uno spazio da qualche parte, il valore LENGTH di Nome diventa la posizione finale di SUBSTR, così viene visualizzato per intero l’ultimo nome.

Questo comando è simile al comando DECODE utilizzato per estrarre il nome; l’unica differenza è che il valore 99 impiegato in quel caso è stato sostituito da LENGTH(Nome), che funziona sempre, mentre 99 potrebbe fallire per un nome con più di 99 caratteri. In questo caso è irrilevante, ma in altri impieghi di DECODE e SUBSTR potrebbe essere importante:

```
select SUBSTR(Nome,
    INSTR(Nome, ' ', -1)+1,
    DECODE(INSTR(Nome, ' '), 0, 0, LENGTH(Nome)))
from NOME;
-----
SUBSTR(NOME,INSTR(NOME,' ')
-----
NELSON
MEDICIS
JOSEPHUS
GAMMIERE
```

Anche questa funzione DECODE potrebbe essere sostituita da GREATEST:

```
select SUBSTR(Nome,
    INSTR(Nome, ' ', -1)+1,
    GREATEST(INSTR(Nome, ' '), 0))
from NOME;
```

Un terzo metodo per ottenere lo stesso risultato utilizza RTRIM. Si ricordi che questa funzione elimina tutto quello che è specificato in *set* dalla parte destra di una stringa, finché non incontra un carattere che non si trova in *set*. In questo caso, RTRIM cancella efficacemente tutte le lettere a destra finché non incontra il primo spazio (proprio prima del cognome) o finché non raggiunge l’inizio della stringa:

```
select
SUBSTR(Nome,LENGTH(RTRIM(NOME, 'ABCDEFGHIJKLMNPQRSTUVWXYZ'))+1)
from NOME;
-----
SUBSTR(NOME,LENGTH(RTRIM(
-----
NELSON
MEDICIS
JOSEPHUS
GAMMIERE
```

`LENGTH` misura la stringa risultante (con il cognome cancellato). In questo modo, si ottiene la posizione dello spazio prima del cognome. Quindi, aggiungendo 1 a questo numero e assegnandolo come posizione iniziale a `SUBSTR` si ottiene proprio il cognome. Si provi a creare una tabella con le colonne Nome e Cognome (tutti i dettagli per la creazione delle tabelle sono riportati nel Capitolo 18):

```
create table DUENOMI(
Nome VARCHAR2(25),
Cognome VARCHAR2(25)
);
```

Table created.

Ora, per caricare i dati della tabella con i nomi e cognomi della tabella NOME si deve utilizzare `insert` con una subquery:

```
insert into DUENOMI (Nome, Cognome)
select
SUBSTR(Nome,1,DECODE(INSTR(Nome, ' '),0,99,INSTR(Nome, ' '))),
SUBSTR(Nome,LENGTH(RTRIM(NOME, 'ABCDEFGHIJKLMNPQRSTUVWXYZ'))+1)
from NOME;
```

Quindi si può controllare il contenuto della tabella DUENOMI:

```
select * from DUENOMI;
```

NOME	COGNOME
HORATIO	NELSON
VALDO	MEDICIS
MARIE	JOSEPHUS
FLAVIUS	GAMMIERE
EDYTHE	

Si possono utilizzare tecniche simili per estrarre le iniziali, e applicare questi metodi anche altrove, per esempio per indirizzi, descrizioni di prodotti, nomi di aziende e così via.

Quando si spostano dei dati da un vecchio sistema a uno nuovo, spesso occorre eseguire una riformattazione piuttosto complessa. In SQL esistono tuttavia delle funzionalità che richiedono un'esperienza di come operano le funzioni e un po' di attenzione per evitare problemi come quelli mostrati qui.

## 16.5 Conteggio delle occorrenze di stringhe in stringhe più lunghe

Si può utilizzare una combinazione delle funzioni `LENGTH` e `REPLACE` per stabilire quante volte una stringa (come 'ABC') è presente in una stringa più lunga (come 'ABCDEFABC'). La funzione `REPLACE` sostituisce uno o più caratteri in una stringa con zero o più caratteri. Pertanto:

```
REPLACE('ADAH', 'A', 'BLAH')
```

valuta la stringa 'ADAH'. Ovunque trovi una 'A', la sostituisce con la stringa 'BLAH'. Quindi, la funzione mostrata in questo esempio restituisce la stringa 'BLAHDBLAHH'.

Si può utilizzare la funzione REPLACE per eliminare i caratteri dalle stringhe. Per esempio, si può sostituire la stringa di caratteri che si sta cercando con un dei valori NULL. Pertanto,

```
REPLACE('GEORGE', 'GE', NULL)
```

restituisce 'OR'. Le due occorrenze separate di 'GE' in 'GEORGE' sono state tutte impostate ciascuna a NULL.

Si può utilizzare la funzione REPLACE per stabilire quante volte una stringa (come 'GE') compare in una stringa più lunga (come 'GEORGE'). Innanzitutto è necessario sostituire la stringa con valori NULL:

```
select REPLACE('GEORGE', 'GE', NULL)
  from DUAL;
```

Il risultato di questa query è 'OR'.

```
RE
--
OR
```

Più importante, è la lunghezza del risultato di questa query che fornisce informazioni su quante volte la stringa è stata trovata. La query seguente calcola la lunghezza della stringa risultante:

```
select LENGTH(REPLACE('GEORGE', 'GE', NULL))
  from DUAL;LENGTH(REPLACE('GEORGE', 'GE',NULL))
-----
2
```

Ora, è possibile calcolare quante volte è stata trovata la stringa. Se si sottrae la lunghezza della stringa accorciata da quella della stringa originaria e si divide questa differenza per la lunghezza della stringa ricercata, si ottiene come risultato il numero di occorrenze della stringa ricercata:

```
select (LENGTH('GEORGE')
       - LENGTH(REPLACE('GEORGE', 'GE', NULL)) )
      / LENGTH('GE') Contatore
  from DUAL;
CONTATORE
-----
2
```

Questo esempio utilizza stringhe al posto di colonne di caratteri per rendere più semplice la comprensione; nelle applicazioni reali si dovrebbe sostituire la stringa originale ('GEORGE') con il nome di colonna. La lunghezza della stringa 'GEORGE' è di sei caratteri. La lunghezza di questa stringa dopo che 'GE' è stato sostituito con NULL è di due caratteri. Pertanto, la differenza di lunghezza tra la stringa originale e quella accorciata è di quattro caratteri. Dividendo i quattro caratteri per la lunghezza della stringa ricercata (due caratteri) si deduce che la stringa è stata trovata due volte.

Il solo problema di questo metodo si presenta quando la stringa ricercata è di zero caratteri (dal momento che non è possibile dividere per zero). La ricerca di una stringa a lunghezza zero può indicare un problema nella logica dell'applicazione che ha inizializzato la ricerca stessa.

## 16.6 Ulteriori informazioni sulle variabili

Il comando `accept` induce SQLPLUS a definire la variabile uguale al valore fornito; questo può avvenire in vari modi: restituendo un messaggio di testo, controllando il tipo di dati fornito e persino evitando la visualizzazione della risposta (come per le password; per ulteriori informazioni si può cercare la voce `accept` nel Capitolo 42).

È possibile passare argomenti al file di avvio, mentre viene avviato, incorporando variabili numerate nelle istruzioni `select` (invece che variabili con nomi).

Per selezionare tutti i record di controllo tra una data e l'altra, si potrebbe scrivere un'istruzione `select` simile a questa:

```
select * from BIBLIOTECA_PRESTITO
where DataPrestito BETWEEN '&1' AND '&2';
```

Questa query può essere salvata come un file (come `checkout.sql`) e avviata dall'interno di SQLPLUS:

```
start checkout.sql 01-JAN-02 31-JAN-02
```

Il file di avvio `checkout.sql` inizierebbe con 01-JAN-02 sostituito a `&1` e 31-JAN-02 sostituito a `&2`. Come per le altre variabili, i tipi di dati DATE e carattere devono essere racchiusi tra apici nell'istruzione SQL. Una limitazione riguarda ogni argomento che segue la parola `start`: dovrà essere costituito da un'unica parola senza spazi.

Le sostituzioni di variabili non sono limitate alle istruzioni SQL. Il file di avvio può utilizzare variabili anche per i comandi SQLPLUS.

Le variabili possono essere concatenate semplicemente ponendole una a fianco dell'altra. Si può concatenare una variabile con una costante utilizzando un punto.

```
select * from BIBLIOTECA_PRESTITO
where DataPrestito BETWEEN '01-&StartMonth.-02' AND '01-&EndMonth.-02';
```

Questa istruzione `select` effettua la query per un mese iniziale e per un mese finale, quindi costruisce le due date utilizzando il punto come operatore di concatenazione.

**NOTA** *Non è necessario alcun punto prima della variabile, ma solo dopo. L'inizio della variabile è comunicato a SQLPLUS dal simbolo &.*

### Comandi set correlati

Normalmente la ‘e’ commerciale (`&`) denota una variabile. Questo simbolo può essere cambiato con un’istruzione `set define` seguita dal simbolo che si preferisce per indicare una variabile.

`set escape` definisce un carattere che può essere posto immediatamente davanti alla ‘e’ commerciale (o a un altro simbolo definito) in modo che SQLPLUS consideri questo simbolo come un letterale e non lo usi per indicare una variabile.

`set concat` può sostituire l’operatore di concatenazione predefinito per le variabili, che è il punto (`.`), con un altro simbolo di una sola lettera. Questa concatenazione di variabili si utilizza in aggiunta, ma separatamente, all’operatore di concatenazione SQL, che in genere è costituito da due barre verticali (`||`).

`set scan` disattiva e attiva la sostituzione di variabili per l’istruzione SQL. Se `scan` è disattivato, qualsiasi variabile nell’istruzione `select` è considerata come un letterale; per esempio, `&Test`

viene trattato come la parola letterale &Test e non come il valore in essa definito. Tuttavia, le variabili nei comandi SQLPLUS vengono ancora sostituite come prima.

A partire da Oracle9i, l'uso di set scan on/off è disapprovato, quindi si dovrà utilizzare set define al posto di set scan.

Per ulteriori opzioni di configurazione ambientale, si può consultare la voce relativa al comando set nel Capitolo 42.

## Capitolo 17

# **DECODE e CASE: if, then ed else in SQL**

- 17.1 **f, then, else**
- 17.2 **Sostituzione di valori con DECODE**
- 17.3 **DECODE in DECODE**
- 17.4 **Greater Than e Less Than in DECODE**
- 17.5 **Uso di CASE**

**L**a funzione DECODE è senza dubbio una delle più potenti nel linguaggio SQL di Oracle. Si tratta di una delle varie estensioni aggiunte da Oracle al linguaggio SQL standard. Questo capitolo analizzerà diversi modi di impiegare DECODE, tra cui la creazione di report a “campi incrociati”. A partire da Oracle9i sono disponibili anche le funzioni CASE e COALESCE che servono per eseguire test logici complessi nelle istruzioni SQL.

Questo è un capitolo complesso e l’argomento è difficile. Gran parte degli argomenti illustrati non sono necessari per la grande maggioranza dei report e possono essere tralasciati se vanno al di là delle proprie esigenze. Si tratta tuttavia di materiale importante nel caso in cui occorra effettuare complesse operazioni di realizzazione di report e di programmazione.

### **17.1 if, then, else**

In programmazione e in logica una comune costruzione di un problema è basata sul modello *if, then, else*. Per esempio, se (*if*) questo giorno è sabato, allora (*then*) Adah gioca a casa; se (*if*) questo giorno è domenica, allora (*then*) Adah va a casa dei nonni; se (*if*) questo giorno è festa, allora (*then*) Adah va dalla zia Dora; altrimenti (*else*) Adah va a scuola. In ogni caso si controlla “questo giorno” e se corrisponde a uno dei giorni elencati, ne consegue un certo risultato; se non corrisponde a nessuno di questi giorni, si ottiene un altro risultato. DECODE rispecchia questo tipo di logica. Nel Capitolo 10 è stata fornita un’introduzione, nella quale è stata mostrata la struttura di base e l’utilizzo di DECODE.

La sintassi di DECODE è la seguente:

```
DECODE(valore,if1,then1,if2,then2,if3,then3,...,else)
```

*valore* rappresenta qualsiasi colonna di una tabella (indipendentemente dal tipo di dati) o qualsiasi risultato di un calcolo, come una data meno un’altra, una funzione SUBSTR su una colonna di caratteri, un numero moltiplicato per un altro e così via. *valore* viene controllato in ogni riga: se è uguale a *if1*, il risultato di DECODE è *then1*, se è uguale a *if2*, il risultato è *then2*, e così via per tutte le coppie *if/then* che si possono costruire. Se *valore* non è uguale a nessuno degli *if*, il risultato è *else*. Ciascuno degli *if, then ed else* può essere una colonna o il risultato di una funzione o di un calcolo. Tra parentesi è possibile racchiudere fino a 255 elementi.

Si consideri la cronologia dei prestiti per i libri della biblioteca, così come è stata registrata nella tabella BIBLIOTECA\_PRESTITO:

```
column Nome format a16
column Titolo format a24 word_wrapped
set pagesize 60
```

```
select * from BIBLIOTECA_PRESTITO;
```

NOME	TITOLO	DATAPREST	DATAREST
JED HOPKINS	INNUMERACY	01-JAN-02	22-JAN-02
GERHARDT KENTGEN	WONDERFUL LIFE	02-JAN-02	02-FEB-02
DORAH TALBOT	EITHER/OR	02-JAN-02	10-JAN-02
EMILY TALBOT	ANNE OF GREEN GABLES	02-JAN-02	20-JAN-02
PAT LAVAY	THE SHIPPING NEWS	02-JAN-02	12-JAN-02
ROLAND BRANDT	THE SHIPPING NEWS	12-JAN-02	12-MAR-02
ROLAND BRANDT	THE DISCOVERERS	12-JAN-02	01-MAR-02
ROLAND BRANDT	WEST WITH THE NIGHT	12-JAN-02	01-MAR-02
EMILY TALBOT	MIDNIGHT MAGIC	20-JAN-02	03-FEB-02
EMILY TALBOT	HARRY POTTER AND THE GOBLET OF FIRE	03-FEB-02	14-FEB-02
PAT LAVAY	THE MISMEASURE OF MAN	12-JAN-02	12-FEB-02
DORAH TALBOT	POLAR EXPRESS	01-FEB-02	15-FEB-02
DORAH TALBOT	GOOD DOG, CARL	01-FEB-02	15-FEB-02
GERHARDT KENTGEN	THE MISMEASURE OF MAN	13-FEB-02	05-MAR-02
FRED FULLER	JOHN ADAMS	01-FEB-02	01-MAR-02
FRED FULLER	TRUMAN	01-MAR-02	20-MAR-02
JED HOPKINS	TO KILL A MOCKINGBIRD	15-FEB-02	01-MAR-02
DORAH TALBOT	MY LEDGER	15-FEB-02	03-MAR-02
GERHARDT KENTGEN	MIDNIGHT MAGIC	05-FEB-02	10-FEB-02

Esaminando questa lista di controllo, si noterà che alcuni libri sono stati presi in prestito per un periodo di tempo abbastanza lungo. Per evidenziare i lettori che hanno tenuto i libri più a lungo, si può ordinare questo elenco in base al numero dei giorni in cui una persona ha tenuto il libro:

```
select Nome, Titolo, DataRestituzione-DataPrestito GiorniFuori
  from BIBLIOTECA_PRESTITO
order by GiorniFuori desc;
```

NOME	TITOLO	GIORNIFUORI
ROLAND BRANDT	THE SHIPPING NEWS	59
ROLAND BRANDT	THE DISCOVERERS	48
ROLAND BRANDT	WEST WITH THE NIGHT	48
GERHARDT KENTGEN	WONDERFUL LIFE	31
PAT LAVAY	THE MISMEASURE OF MAN	31
FRED FULLER	JOHN ADAMS	28
JED HOPKINS	INNUMERACY	21
GERHARDT KENTGEN	THE MISMEASURE OF MAN	20
FRED FULLER	TRUMAN	19
EMILY TALBOT	ANNE OF GREEN GABLES	18
DORAH TALBOT	MY LEDGER	16
EMILY TALBOT	MIDNIGHT MAGIC	14
DORAH TALBOT	POLAR EXPRESS	14
DORAH TALBOT	GOOD DOG, CARL	14

---

JED HOPKINS	TO KILL A MOCKINGBIRD	14
EMILY TALBOT	HARRY POTTER AND THE GOBLET OF FIRE	11
PAT LAVAY	THE SHIPPING NEWS	10
DORAH TALBOT	EITHER/OR	8
GERHARDT KENTGEN	MIDNIGHT MAGIC	5

Il problema è rappresentato da alcuni lettori in particolare, oppure da alcuni tipi di libri che per essere letti richiedono più tempo? Ci sono molte variabili coinvolte in questa questione e analizzarle singolarmente potrebbe portare a decisioni sbagliate. Qual è il numero medio di giorni di prestito per i libri di ogni categoria?

```
select B.NomeCategoria,
       MIN(BC.DataRestituzione-BC.DataPrestito) MinOut,
       MAX(BC.DataRestituzione-BC.DataPrestito) MaxOut,
       AVG(BC.DataRestituzione-BC.DataPrestito) AvgOut
  from BIBLIOTECA_PRESTITO BC, BIBLIOTECA B
 where BC.Titolo = B.Titolo
 group by B.NomeCategoria;
```

NOME CATEGORIA	MINOUT	MAXOUT	AVGOUT
NARRADULTI	10	59	27.6666667
SAGGIADULTI	16	48	29.1111111
CONSADULTI	8	8	8
NARRRAGAZZI	5	18	12
ILLUSRAGAZZI	14	14	14

Questo tentativo è più utile, ma non produce una reale analisi dell'impatto nelle varie categorie prodotto dalle diverse persone che hanno in prestito i libri. Per ottenere questo risultato, si potrebbe eseguire un altro raggruppamento, ma sarà più semplice se si crea un report a *campi incrociati*. Il listato seguente genera un report che mostra il prestito minimo, massimo e medio per persona per categoria. Questa query usa la funzione DECODE per eseguire i calcoli.

```
column NomeCategoria format a11

select B.NomeCategoria,
       MAX(DECODE(BC.Nome, 'FRED FULLER',
                  BC.DataRestituzione-BC.DataPrestito,NULL)) MaxFF,
       AVG(DECODE(BC.Nome, 'FRED FULLER',
                  BC.DataRestituzione-BC.DataPrestito,NULL)) AvgFF,
       MAX(DECODE(BC.Nome, 'DORAH TALBOT',
                  BC.DataRestituzione-BC.DataPrestito,NULL)) MaxDT,
       AVG(DECODE(BC.Nome, 'DORAH TALBOT',
                  BC.DataRestituzione-BC.DataPrestito,NULL)) AvgDT,
       MAX(DECODE(BC.Nome, 'GERHARDT KENTGEN',
                  BC.DataRestituzione-BC.DataPrestito,NULL)) MaxGK,
       AVG(DECODE(BC.Nome, 'GERHARDT KENTGEN',
                  BC.DataRestituzione-BC.DataPrestito,NULL)) AvgGK
  from BIBLIOTECA_PRESTITO BC, BIBLIOTECA B
 where BC.Titolo = B.Titolo
 group by B.NomeCategoria;
```

NOMECategoria	MAXFF	AVGFF	MAXDT	AVGDT	MAXGK	AVGGK
NARRADULTI						
SAGGIADULTI	28	23.5	16	16	31	25.5
CONSADULTI			8	8		
NARRRAGAZZI					5	5
ILLUSRAGAZZI			14	14		

Ora l'output mostra le persone che hanno preso i libri in prestito nella parte superiore: il periodo massimo di prestito per Fred Fuller è la colonna MaxFF, per Dorah Talbot è la colonna MaxDT e per Gerhardt Kentgen è la colonna MaxGK. L'output rivela che per la categoria SaggiAdulti il periodo medio di prestito è il più lungo per ciascuna delle persone indicate e che nel caso di Gerhardt Kentgen è significativamente più lungo rispetto alla sua media nelle altre categorie di libri letti.

Come è stato prodotto questo report? Nella query, il raggruppamento è stato fatto per NomeCategoria. La query sulla colonna MaxFF è mostrata di seguito:

```
select B.NomeCategoria,
       MAX(DECODE(BC.Nome, 'FRED FULLER',
                  BC.DataRestituzione-BC.DataPrestito,NULL)) MaxFF,
```

DECODE esegue un controllo di tipo if-then sui dati: se (if) il valore della colonna BC.Nome in una riga è 'FRED FULLER', allora (then) calcola la differenza tra la DataRestituzione e la DataPrestito, altrimenti (else) restituisce un valore NULL. Questo elenco di valori viene quindi valutato, e viene restituito il valore massimo. Un gruppo simile di operazioni restituisce il periodo medio di prestito per Fred Fuller:

```
AVG(DECODE(BC.Nome, 'FRED FULLER',
            BC.DataRestituzione-BC.DataPrestito,NULL)) AvgFF,
```

## 17.2 Sostituzione di valori con DECODE

Nell'ultimo esempio la funzione DECODE è stata utilizzata per restituire dei valori a una condizione: in base al Nome della persona che ha preso il libro in prestito. DECODE può servire anche per sostituire i valori contenuti in un elenco. Per esempio, se si selezionano i nomi delle categorie da BIBLIOTECA si ottiene questo risultato:

```
select distinct NomeCategoria from BIBLIOTECA;
NOMECategoria
-----
NARRADULTI
SAGGIADULTI
CONSADULTI
NARRRAGAZZI
SAGGIRAGAZZI
ILLUSRAGAZZI
```

Per sostituire questi nomi, si potrebbe unire join BIBLIOTECA a CATEGORIA e selezionare CategoriaPrincipale e CategoriaSecondaria dalla tabella CATEGORIA. Se la lista delle categorie è statica, si potrebbe evitare il join con la tabella CATEGORIA ed eseguire la

sostituzione con la funzione DECODE, come mostrato nel listato seguente. Si osservi che DECODE supporta più combinazioni *if-then* in un'unica chiamata.

```
select distinct
    DECODE(NomeCategoria,'NARRADULTI','Narrativa Adulti',
           'SAGGIADULTI','Saggi Adulti',
           'CONSADULTI','Consultazione Adulti',
           'NARRRAGAZZI','Narrativa Ragazzi',
           'SAGGIRAGAZZI','Saggi Ragazzi',
           'ILLUSRAGAZZI','Illustrati Ragazzi',
           NomeCategoria)
from BIBLIOTECA;

DECODE(NOMECATEGORYIA,
-----
Narrativa Adulti
Saggi Adulti
Consultazione Adulti
Narrativa Ragazzi
Saggi Ragazzi
Illustrati Ragazzi
```

In questo caso, i dati sono statici; per i dati volatili, l'impostazione immutabile delle conversioni nel codice dell'applicazione non sarebbe una pratica di programmazione accettabile. La tecnica proposta in questo esempio è utile per i sistemi di elaborazione delle transazioni in cui si cerca di ridurre al minimo il numero di chiamate al database. In questo esempio, per modificare 'SAGGIADULTI' in 'Saggi Adulti' non è necessario accedere al database; la modifica avviene nella chiamata alla funzione DECODE. Si osservi che se vengono trovate altre categorie, la condizione *else* della funzione DECODE restituisce il valore della colonna CategoriaNome originale.

### 17.3 DECODE in DECODE

È possibile decodificare le chiamate alla funzione DECODE all'interno di altre chiamate alla funzione DECODE. Si supponga di voler far pagare una mora per il ritardo nella restituzione, con ammontare differente per le varie categorie di libri. I libri della categoria adulti possono essere tenuti più a lungo, ma la multa per i giorni di ritardo sarà più salata.

Si parta da una tariffa uniforme di base pari a • 0.20 al giorno per i libri che vengono tenuti più di 14 giorni:

```
column Nome format a16
column Titolo format a20 word_wrapped
column GiorniPrestito format 999.99 heading 'Giorni Prestito'
column GiorniRitardo format 999.99 heading 'Giorni Ritardo'
set pagesize 60
break on Nome

select Nome, Titolo, DataRestituzione,
       DataRestituzione-DataPrestito as GiorniPrestito /*Conta i giorni*/,
       DataRestituzione-DataPrestito -14 GiorniRitardo,
       (DataRestituzione-DataPrestito -14)*0.20 TassaRitardo
```

```
from BIBLIOTECA_PRESTITO
where DataRestituzione-DataPrestito > 14
order by Nome, DataPrestito;
```

NOME	TITOLO	DATAREST	Giorni Prestito	Giorni Ritardo	TASSARITARDO
DORAH TALBOT	MY LEDGER	03-MAR-02	16.00	2.00	.4
EMILY TALBOT	ANNE OF GREEN GABLES	20-JAN-02	18.00	4.00	.8
FRED FULLER	JOHN ADAMS	01-MAR-02	28.00	14.00	2.8
	TRUMAN	20-MAR-02	19.00	5.00	1
GERHARDT KENTGEN	WONDERFUL LIFE	02-FEB-02	31.00	17.00	3.4
	THE MISMEASURE OF	05-MAR-02	20.00	6.00	1.2
	MAN				
JED HOPKINS	INNUMERACY	22-JAN-02	21.00	7.00	1.4
PAT LAVAY	THE MISMEASURE OF	12-FEB-02	31.00	17.00	3.4
	MAN				
ROLAND BRANDT	THE SHIPPING NEWS	12-MAR-02	59.00	45.00	9
	THE DISCOVERERS	01-MAR-02	48.00	34.00	6.8
	WEST WITH THE NIGHT	01-MAR-02	48.00	34.00	6.8

Per i libri delle categorie Adulti si deve aumentare il numero dei giorni di ritardo consentiti a 21. In questo modo, non cambierà il numero dei giorni di prestito, ma soltanto il calcolo nella colonna GiorniRitardo. Dato che NomeCategoria non è una colonna delle tabella BIBLIOTECA\_PRESTITO, questa modifica richiede anche l'aggiunta della tabella BIBLIOTECA alla clausola from.

```
select BC.Nome, BC.Titolo, BC.DataRestituzione,
       BC.DataRestituzione-BC.DataPrestito as GiorniFuori /*Conta i giorni*/,
       DECODE(SUBSTR(NomeCategoria,-1,5), 'ADULTI',
              BC.DataRestituzione-BC.DataPrestito -21,
              BC.DataRestituzione-BC.DataPrestito -14 ) GiorniRitardo,
       DECODE(SUBSTR(NomeCategoria,-1,5), 'ADULTI',
              (BC.DataRestituzione-BC.DataPrestito -21)*0.30,
              (BC.DataRestituzione-BC.DataPrestito -14)*0.20 ) TassaRitardo
  from BIBLIOTECA_PRESTITO BC, BIBLIOTECA B
 where BC.Titolo = B.Titolo
   and BC.DataRestituzione-BC.DataPrestito
        DECODE(SUBSTR(NomeCategoria,6), 'ADULTI',21,14)
 order by BC.Nome, BC.DataPrestito;
```

NOME	TITOLO	DATAREST	Giorni Prestito	Giorni Ritardo	TASSARITARDO
EMILY TALBOT	ANNE OF GREEN GABLES	20-JAN-02	18.00	4.00	.8
FRED FULLER	JOHN ADAMS	01-MAR-02	28.00	7.00	2.1
GERHARDT KENTGEN	WONDERFUL LIFE	02-FEB-02	31.00	10.00	3
PAT LAVAY	THE MISMEASURE OF	12-FEB-02	31.00	10.00	3
	MAN				
ROLAND BRANDT	THE SHIPPING NEWS	12-MAR-02	59.00	38.00	11.4
	THE DISCOVERERS	01-MAR-02	48.00	27.00	8.1
	WEST WITH THE NIGHT	01-MAR-02	48.00	27.00	8.1

Nella clausola select, la logica della query per la colonna GiorniRitardo è la seguente:

```
DECODE(SUBSTR(NomeCategoria,1,5), 'ADULT',
       BC.DataRestituzione-BC.DataPrestito -21,
       BC.DataRestituzione-BC.DataPrestito -14 ) GiorniRitardo
```

Nel formato if-then-else di DECODE, questo significa che “Se gli ultimi sei caratteri della colonna NomeCategoria corrispondono alla stringa “ADULTI”, allora si deve sottrarre 21 dal numero dei giorni di ritardo; altrimenti, si deve sottrarre 14”. Il calcolo della colonna TassaRitardo utilizza una logica simile. La clausola where usa DECODE per stabilire il valore di limitazione per le righe restituite: per i libri della categoria ADULTI, il numero concesso è 21; altrimenti, 14:

```
and BC.DataRestituzione-BC.DataPrestito >
    DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',21,14)
```

Ora, si aggiunga un’altra regola: per i libri della categoria Narrativa adulti, la mora di ritardo sarà esattamente il doppio di quella prevista per gli altri libri per adulti. Dall’ultima query, il calcolo per TassaRitardo è:

```
DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',
       (BC.DataRestituzione-BC.DataPrestito -21)*0.30,
       (BC.DataRestituzione-BC.DataPrestito -14)*0.20 ) TassaRitardo
```

La nuova regola richiede una verifica di categoria ulteriore nel comando DECODE già eseguito. Di seguito viene mostrato il nuovo calcolo per TassaRitardo:

```
DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',
       DECODE(SUBSTR(NomeCategoria,-7,4), 'NARR',
              (BC.DataRestituzione-BC.DataPrestito -21)*0.60,
              (BC.DataRestituzione-BC.DataPrestito -21)*0.30),
              (BC.DataRestituzione-BC.DataPrestito -14)*0.20 ) TassaRitardo
```

Esaminando il listato precedente, la prima funzione DECODE indica a Oracle che se le ultime sei lettere di NomeCategoria sono ‘ADULTI’, allora si deve eseguire la seconda funzione DECODE. La seconda funzione DECODE comunica a Oracle che se la settima, l’ottava, la nona e la decima lettera di NomeCategoria a partire dalla fine sono “NARR”, allora la mora di ritardo ammonta a €0,60 per giorno dopo il ventunesimo giorno; altrimenti, ammonterà a €0,30 per giorno dopo il ventunesimo. A questo punto, la funzione DECODE interna viene completata. Per la prima funzione DECODE, la clausola else (per i libri non della categoria ADULTI), specifica il calcolo della tassa di ritardo. Nel listato seguente sono mostrati la query e i risultati; è possibile vedere l’impatto mettendo a confronto la colonna TassaRitardo per ‘THE SHIPPING NEWS’ di questo report con quella dell’ultimo report.

```
select BC.Nome, BC.Titolo, BC.DataRestituzione,
       BC.DataRestituzione-BC.DataPrestito as GiorniFuori /*Conta i giorni*/,
       DECODE(SUBSTR(NomeCategoria,-1,5), 'ADULTI',
              BC.DataRestituzione-BC.DataPrestito -21,
              BC.DataRestituzione-BC.DataPrestito -14 ) GiorniRitardo,
       DECODE(SUBSTR(NomeCategoria,-1,5), 'ADULTI',
              DECODE(SUBSTR(NomeCategoria,-7,4), 'NARR',
                     (BC.DataRestituzione-BC.DataPrestito -21)*0.60,
                     (BC.DataRestituzione-BC.DataPrestito -21)*0.30),
                     (BC.DataRestituzione-BC.DataPrestito -14)*0.20 ) TassaRitardo
  from BIBLIOTECA_PRESTITO BC, BIBLIOTECA B
 where BC.Titolo = B.Titolo
   and BC.DataRestituzione-BC.DataPrestito >
        DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',21,14)
 order by BC.Nome, BC.DataPrestito;
```

NOME	TITOLO	DATAREST	Giorni Prestito	Giorni Ritardo	TASSARITARDO
EMILY TALBOT	ANNE OF GREEN GABLES	20-JAN-02	18.00	4.00	.8
FRED FULLER	JOHN ADAMS	01-MAR-02	28.00	7.00	2.1
GERHARDT KENTGEN	WONDERFUL LIFE	02-FEB-02	31.00	10.00	3
PAT LAVAY	THE MISMEASURE OF MAN	12-FEB-02	31.00	10.00	3
ROLAND BRANDT	THE SHIPPING NEWS	12-MAR-02	59.00	38.00	22.8
	THE DISCOVERERS	01-MAR-02	48.00	27.00	8.1
	WEST WITH THE NIGHT	01-MAR-02	48.00	27.00	8.1

È possibile annidare funzioni DECODE in altre funzioni DECODE per supportare la logica complessa nei processi di elaborazione dei dati. Per esempio, si potrebbe decidere di visualizzare una colonna se ha un valore non NULL, e una seconda colonna se la prima è NULL. Con DECODE, ovvero una semplice chiamata di funzione:

```
DECODE(Colonna1, NULL, Colonna2, Colonna1)
```

se Colonna1 è NULL, viene restituita Colonna2; altrimenti, viene restituito il valore non NULL di Colonna1. Per eseguire una logica simile, si può usare anche NVL o le funzioni COALESCE e NULLIF di Oracle9i.

COALESCE restituisce il primo valore non NULL che incontra in un elenco di valori. L'ultimo esempio di DECODE potrebbe essere riscritto in questo modo:

```
COALESCE(Colonna1, Colonna2)
```

Dato che COALESCE è una funzione nuova e il suo nome non ne rivela chiaramente l'impiego, ci si dovrà accertare di aver inserito dei commenti nel codice per spiegare i test logici effettuati.

## 17.4 Greater Than e Less Than in DECODE

Decode supporta controlli logici, ma in che modo si possono effettuare confronti numerici in questo formato? Spesso la soluzione più semplice consiste nell'usare la funzione SIGN, che restituisce 1 se il numero è positivo, 0 se è zero e -1 se il numero è negativo. Dato che SIGN opera sui numeri, si possono valutare quelle funzioni che restituiscono un numero, compresa l'aritmetica delle date.

Si provi a modificare nuovamente la regola per TassaRitardo. Con lo stesso calcolo di base, si potrà cambiare il risultato in modo da evitare che vengano raccolte le more di ritardo il cui importo è minore o uguale a €4,00. Ecco il calcolo di TassaRitardo originale:

```
DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',
       DECODE(SUBSTR(NomeCategoria,-7,4), 'NARR',
              (BC. DataRestituzione-BC.DataPrestito -21)*0.60,
              (BC. DataRestituzione-BC.DataPrestito -21)*0.30),
              (BC. DataRestituzione-BC.DataPrestito -14)*0.20 ) TassaRitardo
```

Se questo valore è minore di 4, si otterrà 0; altrimenti, si otterrà il valore calcolato. Per implementare questo requisito, si deve sottrarre 4 dal valore di TassaRitardo; se il risultato è positivo (ossia il valore di SIGN è 1), allora si dovrà restituire questo risultato; altrimenti 0.

```

DECODE(SIGN(
    DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',
           DECODE(SUBSTR(NomeCategoria,-7,4), 'NARR',
                  (BC.DataRestituzione-BC.DataPrestito -21)*0.60,
                  (BC.DataRestituzione-BC.DataPrestito -21)*0.30),
                  (BC.DataRestituzione-BC.DataPrestito -14)*0.20 ) -4),
    1,
    DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',
           DECODE(SUBSTR(NomeCategoria,-7,4), 'NARR',
                  (BC.DataRestituzione-BC.DataPrestito -21)*0.60,
                  (BC.DataRestituzione-BC.DataPrestito -21)*0.30),
                  (BC.DataRestituzione-BC.DataPrestito -14)*0.20 ),
    0) TassaRitardo
)

```

Partendo da una base semplice, questa serie di chiamate alle funzioni DECODE consente di applicare una logica molto complessa nei calcoli di TassaRitardo. La prima funzione DECODE valuta il segno del risultato della funzione DECODE successiva quando a esso viene sottratto 4. Se questo numero è positivo, viene restituita la tassa di ritardo calcolata; altrimenti, viene restituito 0. Il listato seguente mostra questo calcolo e l'output ottenuto.

```

select BC.Nome, BC.Titolo, BC.DataRestituzione,
       BC.DataRestituzione-BC.DataPrestito as GiorniFuori /*Conta i giorni*/,
       DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',
              BC.DataRestituzione-BC.DataPrestito -21,
              BC.DataRestituzione-BC.DataPrestito -14 ) GiorniRitardo,
DECODE(SIGN(
    DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',
           DECODE(SUBSTR(NomeCategoria,-7,4), 'NARR',
                  (BC.DataRestituzione-BC.DataPrestito -21)*0.60,
                  (BC.DataRestituzione-BC.DataPrestito -21)*0.30),
                  (BC.DataRestituzione-BC.DataPrestito -14)*0.20 ) -4),
    1,
    DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',
           DECODE(SUBSTR(NomeCategoria,-7,4), 'NARR',
                  (BC.DataRestituzione-BC.DataPrestito -21)*0.60,
                  (BC.DataRestituzione-BC.DataPrestito -21)*0.30),
                  (BC.DataRestituzione-BC.DataPrestito -14)*0.20 ),
    0) TassaRitardo
)
from BIBLIOTECA_PRESTITO BC, BIBLIOTECA B
where BC.Titolo = B.Titolo
and BC.DataRestituzione-BC.DataPrestito >
      DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',21,14)
order by BC.Nome, BC.DataPrestito;

```

NOME	TITOLO	DATAREST	Giorni Prestito	Giorni Ritardo	TASSARITARDO
EMILY TALBOT	ANNE OF GREEN GABLES	20-JAN-02	18.00	4.00	0
FRED FULLER	JOHN ADAMS	01-MAR-02	28.00	7.00	0
GERHARDT KENTGEN	WONDERFUL LIFE	02-FEB-02	31.00	10.00	0
PAT LAVAY	THE MISMEASURE OF MAN	12-FEB-02	31.00	10.00	0
ROLAND BRANDT	THE SHIPPING NEWS	12-MAR-02	59.00	38.00	22.8
	THE DISCOVERERS	01-MAR-02	48.00	27.00	8.1
	WEST WITH THE NIGHT	01-MAR-02	48.00	27.00	8.1

È possibile eliminare la visualizzazione delle prime quattro righe (con le tasse di ritardo pari a €0) modificando in modo analogo la clausola where.

## 17.5 Uso di CASE

A partire da Oracle9i, al posto della funzione DECODE è possibile usare la funzione CASE. Questa funzione usa le parole chiave when, then, else ed end per indicare il percorso logico seguito, che potrebbe semplificare il codice risultante rispetto a quello di una equivalente DECODE.

Si consideri questo semplice esempio di DECODE tratto da un paragrafo precedente di questo capitolo:

```
select distinct
    DECODE(NomeCategoria,'NARRADULTI','Narrativa Adulti',
           'SAGGIADULTI','Saggi Adulti',
           'CONSADULTI','Consultazione Adulti',
           'NARRRAGAZZI','Narrativa Ragazzi',
           'SAGGIRAGAZZI','Saggi Ragazzi',
           'ILLUSRAGAZZI','Illustrati Ragazzi',
           NomeCategoria)
  from BIBLIOTECA;
```

La funzione CASE equivalente è:

```
select distinct
    CASE NomeCategoria
        when 'NARRADULTI' then 'Narrativa Adulti'
        when 'SAGGIADULTI' then 'Saggi Adulti'
        when 'CONSADULTI' then 'Consultazione Adulti'
        when 'NARRRAGAZZI' then 'Narrativa Ragazzi'
        when 'SAGGIRAGAZZI' then 'Saggi Ragazzi'
        when 'ILLUSRAGAZZI' then 'Illustrati Ragazzi'
        else NomeCategoria
    end
  from BIBLIOTECA;
CASECATEGORYNAMEWHEN
-----
Narrativa Adulti
Saggi Adulti
Consultazione Adulti
Narrativa Ragazzi
Saggi Ragazzi
Illustrati Ragazzi
```

CASE valuta la prima clausola when e restituisce una corrispondenza se la condizione di limitazione è stata rispettata. Come per DECODE, è possibile annidare le chiamate alle funzioni CASE. Come si potrebbero eseguire i calcoli della colonna TassaRitardo con CASE?

La funzione DECODE aveva iniziato calcolando un tasso più alto per i libri della categoria ADULTI:

```
DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',
       (BC.DataRestituzione-BC.DataPrestito -21)*0.30,
       (BC.DataRestituzione-BC.DataPrestito -14)*0.20 ) TassaRitardo
```

La funzione CASE equivalente è:

```
CASE SUBSTR(NomeCategoria,-1,6)
when 'ADULTI' then (BC.DataRestituzione-BC.DataPrestito -21)*0.30
else (BC.DataRestituzione-BC.DataPrestito -14)*0.20
end
```

La seconda regola (i libri della categoria NARRADULTI hanno tasse di ritardo doppie) richiede un comando CASE annidato:

```
CASE SUBSTR(NomeCategoria,-1,6)
when 'ADULTI' then
  CASE SUBSTR(NomeCategoria,-7,4)
    when 'NARR' then (BC.DataRestituzione-BC.DataPrestito -21)*0.60
    else (BC.DataRestituzione-BC.DataPrestito -21)*0.30
    end
  else (BC.DataRestituzione-BC.DataPrestito -14)*0.20
end
```

Questo risultato è più complesso, ma seguire la logica è davvero molto semplice, e in genere è anche più facile da gestire rispetto a quella della clausola DECODE equivalente. Ora, si consideri la condizione finale: se la tassa di ritardo calcolata è minore o uguale a €4, si deve ottenere 0. Dato che si utilizza la funzione CASE, non è necessario ricorrere a una funzione SIGN: è sufficiente usare un confronto “<” durante l’elaborazione del comando. L’esempio seguente mostra come aggiungere questo controllo: viene aggiunta un’altra funzione CASE, e la chiamata alla funzione CASE interna viene racchiusa tra parentesi. Il risultato viene confrontato con \$4:

```
CASE when
  (CASE SUBSTR(NomeCategoria,-1,6)
  when 'ADULTI' then
    CASE SUBSTR(NomeCategoria,-7,4)
      when 'NARR' then (BC.DataRestituzione-BC.DataPrestito -21)*0.60
      else (BC.DataRestituzione-BC.DataPrestito -21)*0.30
      end
    else (BC.DataRestituzione-BC.DataPrestito -14)*0.20
  end)
  < 4 then 0
```

La clausola else per questa chiamata alla funzione CASE è la stessa per l’esecuzione della funzione CASE interna: calcola la tassa del ritardo.

```
else
  CASE SUBSTR(NomeCategoria,-1,6)
  when 'ADULTI' then
    CASE SUBSTR(NomeCategoria,-7,4)
      when 'NARR' then (BC.DataRestituzione-BC.DataPrestito -21)*0.60
      else (BC.DataRestituzione-BC.DataPrestito -21)*0.30
      end
    else (BC.DataRestituzione-BC.DataPrestito -14)*0.20
  end
end
```

Nel listato seguente è mostrata la query completa, insieme all'output.

```

column TassaRitardo format 999.99

select BC.Nome, BC.Titolo, BC.DataRestituzione,
       BC.DataRestituzione-BC.DataPrestito GiorniFuori /*Conta i giorni*/,
       DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',
              BC.DataRestituzione-BC.DataPrestito -21,
              BC.DataRestituzione-BC.DataPrestito -14 ) GiorniRitardo,
CASE when
  (CASE SUBSTR(NomeCategoria,-1,6)
   when 'ADULTI' then
     CASE SUBSTR(NomeCategoria,-7,4)
      when 'NARR' then (BC.DataRestituzione-BC.DataPrestito -21)*0.60
      else (BC.DataRestituzione-BC.DataPrestito -21)*0.30
      end
     else (BC.DataRestituzione-BC.DataPrestito -14)*0.20
    end)
   < 4 then 0 else
  CASE SUBSTR(NomeCategoria,-1,6)
   when 'ADULTI' then
     CASE SUBSTR(NomeCategoria,-7,4)
      when 'NARR' then (BC.DataRestituzione-BC.DataPrestito -21)*0.60
      else (BC.DataRestituzione-BC.DataPrestito -21)*0.30
      end
     else (BC.DataRestituzione-BC.DataPrestito -14)*0.20
    end
  end
end AS TassaRitardo
from BIBLIOTECA_PRESTITO BC, BIBLIOTECA B
where BC.Titolo = B.Titolo
  and BC.DataRestituzione-BC.DataPrestito >
        DECODE(SUBSTR(NomeCategoria,-1,6), 'ADULTI',21,14)
order by BC.Nome, BC.DataPrestito;
```

NOME	TITOLO	DATAREST	Giorni Prestito	Giorni Ritardo	TASSARITARDO
EMILY TALBOT	ANNE OF GREEN GABLES	20-JAN-02	18.00	4.00	.00
FRED FULLER	JOHN ADAMS	01-MAR-02	28.00	7.00	.00
GERHARDT KENTGEN	WONDERFUL LIFE	02-FEB-02	31.00	10.00	.00
PAT LAVAY	THE MISMEASURE OF MAN	12-FEB-02	31.00	10.00	.00
ROLAND BRANDT	THE SHIPPING NEWS	12-MAR-02	59.00	38.00	22.80
	THE DISCOVERERS	01-MAR-02	48.00	27.00	8.10
	WEST WITH THE NIGHT	01-MAR-02	48.00	27.00	8.10

Se si confronta la versione di CASE con quella di DECODE proposta in precedenza, si potrà notare che il comando CASE ha sei linee in più ma è più semplice da leggere e gestire. Per i database Oracle9i, CASE offre un'alternativa potente a DECODE; inoltre CASE e DECODE mettono entrambe a disposizione soluzioni solide quando nelle query è necessario applicare della logica.

## Capitolo 18

# Creazione, eliminazione e modifica di tabelle e viste

- 18.1 **Creazione di una tabella**
- 18.2 **Eliminazione di tabelle**
- 18.3 **Modifica di tabelle**
- 18.4 **Creazione di una vista**
- 18.5 **Creazione di una tabella a partire da un'altra**
- 18.6 **Creazione di una tabella di solo indice**
- 18.7 **Uso di tabelle partizionate**
- 18.8 **Ridefinizione in linea delle tabelle**

**F**inora si è posta particolare attenzione all'utilizzo delle tabelle. In questo capitolo, si imparerà a creare, eliminare e modificare tabelle, a creare viste e opzioni come tabelle di solo indice o tabelle partizionate. Finora sono stati introdotti diversi comandi `create table`; questo capitolo conferma questi esempi e mostra come utilizzare le opzioni più recenti.

## 18.1 Creazione di una tabella

Si consideri la tabella STRANO. Assomiglia alla tabella COMFORT, analizzata in precedenza nel libro, tuttavia è utilizzata per registrare le città con valori di temperatura insoliti.

```
describe STRANO
```

Nome	Null?	Type
CITTA	NOT NULL	VARCHAR2(13)
DATACAMP	NOT NULL	DATE
MEZZOGIORNO		NUMBER(3,1)
MEZZANOTTE		NUMBER(3,1)
PRECIPITAZIONE		NUMBER

Le colonne della tabella STRANO rappresentano i tre tipi di dati principali in Oracle, VARCHAR2, DATE e NUMBER. Ecco l'istruzione SQL che ha creato questa tabella di Oracle:

```
create table STRANO (
Citta      VARCHAR2(13) NOT NULL,
DataCampione  DATE NOT NULL,
Mezzogiorno  NUMBER(3,1),
Mezzanotte   NUMBER(3,1),
Precipitazione NUMBER
);
```

Di seguito sono elencati gli elementi fondamentali di questo comando:

- Le parole `create table`.
- Il nome della tabella.
- Una parentesi di apertura.
- Le definizioni delle colonne.
- Una parentesi di chiusura.
- Un terminatore SQL.

Le singole definizioni di colonne sono separate da virgole. Non vi è alcuna virgola dopo l'ultima definizione. I nomi delle tabelle e delle colonne devono iniziare con una lettera dell'alfabeto, ma possono comprendere lettere, numeri e trattini di sottolineatura. I nomi possono essere lunghi da 1 a 30 caratteri, devono essere unici all'interno della tabella e non possono essere uguali a una parola riservata di Oracle (consultare la voce "Nomì di oggetto" nel Capitolo 42).

Se i nomi non sono racchiusi tra virgolette, nella creazione di una tabella non si distingue tra lettere maiuscole e minuscole. Non esistono opzioni per i tipi di dati DATE. Per i tipi di dati carattere dev'essere specificata la lunghezza massima. I NUMBER possono avere una precisione elevata (fino a 38 cifre), o una precisione fissa in base al massimo numero di cifre e al numero di posizioni consentite a destra del punto decimale (un campo Importo per la valuta americana, per esempio, dovrebbe avere solo due posizioni decimali).

**NOTA** *Non si devono racchiudere i nomi di tabelle e colonne tra virgolette, altriimenti le maiuscole e le minuscole avrebbero rilevanza, creando gravi problemi agli utenti o agli sviluppatori.*

Per altre opzioni di `create table` relative alle caratteristiche che si riferiscono agli oggetti, fare riferimento alla Parte IV di questo libro.

## Larghezza delle colonne di caratteri e precisione dei dati NUMBER

Specificare una lunghezza massima per le colonne di caratteri (di tipo CHAR e VARCHAR2) e una precisione per le colonne NUMBER può avere conseguenze che devono essere prese in considerazione durante la progettazione di una tabella. Decisioni errate possono essere corrette in seguito con il comando `alter table`, seppure con difficoltà.

### Come decidere la larghezza adeguata

Se una colonna di caratteri non è abbastanza larga per i dati da inserirvi, l'istruzione `insert` fallisce, producendo come risultato questo messaggio di errore:

```
ERROR at line 1: ORA-01401: inserted value too large for column
```

La larghezza massima per le colonne di tipo CHAR (a lunghezza fissa) è di 2.000 caratteri, mentre le colonne (di caratteri a lunghezza variabile) di tipo VARCHAR2 possono contenere un massimo di 4.000 caratteri. Quando si assegna la larghezza a una colonna, occorre riservare spazio sufficiente per tutte le possibilità future. Per esempio, una colonna CHAR(15) per il nome di una città prima o poi creerà dei problemi. Per cui si dovrà modificare la tabella oppure troncare o distorcere i nomi di alcune città.

**NOTA** Non vi è alcuno svantaggio nel definire in Oracle una colonna di VARCHAR2 di grandi dimensioni. Oracle è abbastanza intelligente da non memorizzare spazi vuoti alla fine di queste colonne. Il nome della città "SAN FRANCISCO", per esempio, viene memorizzato in 13 spazi anche se la colonna è stata definita come VARCHAR2(50). Se in una colonna non vi è nulla (NULL), Oracle non vi memorizza alcunché, nemmeno uno spazio vuoto (in realtà memorizza un paio di byte per informazioni di controllo interne al database, ma questo non dipende dalla dimensione specificata per la colonna). L'unico effetto che si ha nella scelta di una larghezza di colonna ampia è nella formattazione predefinita di SQLPLUS per le colonne. Infatti, SQLPLUS crea un'intestazione predefinita di larghezza corrispondente alla definizione di VARCHAR2.

### Scelta della precisione per i numeri

Una colonna di NUMBER con una precisione non corretta può avere diverse conseguenze. Oracle potrebbe respingere il tentativo di inserire la riga di dati o potrebbe ridurre la precisione dei dati stessi. Ecco quattro righe di dati che stanno per essere inserite in Oracle:

```
insert into STRANO values
  ('PLEASANT LAKE', '21-MAR-01',
   39.99, -1.31, 3.6);

insert into STRANO values
  ('PLEASANT LAKE', '22-JUN-01',
   101.44, 86.2, 1.63);

insert into STRANO values
  ('PLEASANT LAKE', '23-SEP-01',
   92.85, 79.6, 1.00003);

insert into STRANO values
  ('PLEASANT LAKE', '22-DEC-01',
   -17.445, -10.4, 2.4);
```

Ed ecco i risultati di questo tentativo:

```
insert into STRANO values ('PLEASANT LAKE',
                           '21-MAR-01', 39.99, -1.31, 3.6);
1 row created.

insert into STRANO values ('PLEASANT LAKE',
                           '22-JUN-01', 101.44, 86.2, 1.63);
                           '22-JUN-01', 101.44, 86.2, 1.63)
                           *
ERROR at line 2:
ORA-01438: value larger than specified precision allows for this column

insert into STRANO values ('PLEASANT LAKE',
                           '23-SEP-01', 92.85, 79.6, 1.00003);
1 row created.

insert into STRANO values ('PLEASANT LAKE',
                           '22-DEC-01', -17.445, -10.4, 2.4);
1 row created.
```

La prima, la terza e la quarta riga sono state inserite, mentre il secondo inserimento non è riuscito perché 101.44 supera la precisione impostata nell'istruzione `create table`, dove Mezzogiorno è stata definita come `NUMBER(3,1)`. 3 indica il numero massimo di cifre memorizzate da Oracle, mentre 1 specifica che una di queste tre cifre è riservata per una posizione a destra del punto decimale. Pertanto 12.3 è un numero lecito, ma 123.4 non lo è.

Si osservi che in questo caso l'errore è stato causato da 101, non da .44, perché `NUMBER(3,1)` lascia disponibili solo due posizioni a sinistra del punto decimale. 44 non produce un errore "value larger than specified precision" (valore troppo grande rispetto alla precisione specificata). Infatti, viene arrotondato semplicemente a una sola cifra decimale. Questo comportamento verrà dimostrato tra breve, tuttavia prima occorre esaminare i risultati di una query delle quattro righe che si è cercato di inserire:

```
select * from STRANO;
```

CITTA	DATA	CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
PLEASANT LAKE	21-MAR-01		4.4	-18.5	3.6
PLEASANT LAKE	23-SEP-01		33.8	26.4	1.00003
PLEASANT LAKE	22-DEC-01		-27.4	-23.5	2.4

Tre righe sono state inserite correttamente; manca solo quella che ha creato problemi. Oracle ha annullato automaticamente l'unica istruzione `insert` che non è riuscita.

## Arrotondamento durante l'inserimento

Se si corregge l'istruzione `create table` e si aumenta il numero di cifre disponibili per Mezzogiorno e Mezzanotte, come mostrato di seguito:

```
drop table STRANO;
create table STRANO (
    Citta      VARCHAR2(13) NOT NULL,
    DataCampione DATE NOT NULL,
    Mezzogiorno NUMBER(4,1),
    Mezzanotte  NUMBER(4,1),
    Precipitazione NUMBER
);
```

le quattro istruzioni `insert` vengono completate con successo. Una query può dimostrarlo:

```
insert into STRANO values
('PLEASANT LAKE','21-MAR-01',
 39.99, -1.31, 3.6);

insert into STRANO values
('PLEASANT LAKE','22-JUN-01',
 101.44, 86.2, 1.63);

insert into STRANO values
('PLEASANT LAKE','23-SEP-01',
 92.85, 79.6, 1.00003);

insert into STRANO values
('PLEASANT LAKE','22-DEC-01',
```

```
-17.445, -10.4, 2.4);
```

```
select * from STRANO;
```

CITTA	DATACAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
PLEASANT LAKE	21-MAR-01	4.4	-18.5	3.6
PLEASANT LAKE	22-JUN-01	38.5	30.1	1.63
PLEASANT LAKE	23-SEP-01	33.8	26.4	1.00003
PLEASANT LAKE	22-DEC-01	-27.4	-23.5	2.4

Si esamini la prima istruzione `insert`. Il valore per Mezzogiorno è 39.99. Nella query è arrotondato a 40. Mezzanotte nell'istruzione `insert` vale -1.31, mentre nella query vale -1.3. Oracle arrotonda il numero in base alla cifra immediatamente a destra della precisione consentita. La Tabella 18.1 mostra gli effetti della precisione in vari esempi. Per vedere alcuni esempi della funzione `ROUND`, si rimanda al Capitolo 8.

## Vincoli in create table

L'istruzione `create table` consente di imporre vari tipi di vincoli su una tabella: chiavi candidate, chiavi primarie, chiavi esterne e condizioni di controllo. Una clausola `constraint` può vincolare una singola colonna o un gruppo di colonne in una tabella. Lo scopo di questi vincoli è fare in modo che Oracle effettui la maggior parte del lavoro per conservare l'integrità del database. Più sono i vincoli aggiunti a una definizione di tabella, minore è il lavoro da svolgere nell'applicazione per la gestione dei dati. Tuttavia, più vincoli vi sono in una tabella, maggiore è il tempo necessario per aggiornare i dati.

Esistono due modi per specificare i vincoli: in una definizione di colonna (un vincolo di *colonna*) o alla fine dell'istruzione `create table` (un vincolo di *tabella*). Le clausole che contengono diverse colonne devono essere vincoli di tabella.

### La chiave candidata

Una *chiave candidata* è una combinazione di una o più colonne, i valori delle quali identificano in modo univoco ciascuna riga di una tabella. Il listato seguente mostra la creazione di un vincolo `UNIQUE` per la tabella `STRANO`:

```
drop table STRANO;
create table STRANO (
Citta      VARCHAR2(13) NOT NULL,
DataCampione DATE NOT NULL,
Mezzogiorno NUMBER(4,1),
Mezzanotte NUMBER(4,1),
Precipitazione NUMBER,
constraint STRANO_UQ UNIQUE (Citta, DataCampione)
);
```

La chiave di questa tabella è la combinazione di Città e DataCampione. Si noti che entrambe le colonne sono dichiarate anche come `NOT NULL`. Questa caratteristica consente di impedire che i dati vengano inseriti nella tabella in cui alcune colonne non hanno alcun valore. Ovviamente, le informazioni sulle temperature e sulle precipitazioni non sono utili senza sapere dove e quando sono state raccolte. Questa tecnica è comune per le colonne che sono chiave primaria di una tabella, ma è utile anche quando determinate colonne sono importanti perché la riga di dati sia significativa. Se non viene specificato `NOT NULL`, la colonna può avere valori `NULL`.

**Tabella 17.1** Esempi degli effetti della precisione sui valori inseriti.

NELL'ISTRUZIONE INSERT	VALORE REALE NELLA TABELLA
<b>Con precisione NUMBER(4,1)</b>	
123.4	123.4
123.44	123.4
123.45	123.5
123.445	123.4
1234.5	insert fallisce
<b>Con precisione NUMBER(4)</b>	
123.4	123
123.44	123
123.45	123
123.445	123
1234.5	1235
12345	insert fallisce
<b>Con precisione NUMBER(4,-1)</b>	
123.4	120
123.44	120
123.45	120
123.445	120
125	130
1234.5	1230
12345	insert fallisce
<b>Con precisione NUMBER</b>	
123.4	123.4
123.44	123.44
123.45	123.45
123.445	123.445
125	125
1234.5	1234.5
12345.6789012345678	12345.6789012345678

## La chiave primaria

La *chiave primaria* di una tabella è una delle chiavi candidate cui vengono assegnate alcune caratteristiche speciali. Può esistere una sola chiave primaria e una colonna di tale chiave non può contenere valori NULL:

```
create table STRANO (
Citta          VARCHAR2(13),
DataCampione   DATE,
Mezzogiorno    NUMBER(4,1),
Mezzanotte     NUMBER(4,1),
Precipitazione NUMBER,
constraint STRANO_PK PRIMARY KEY (Citta, DataCampione)
);
```

Questa istruzione `create table` ha lo stesso effetto di quella precedente, eccetto che si possono avere più vincoli **UNIQUE** ma solo un vincolo **PRIMARY KEY**.

Per chiavi primarie o candidate di una sola colonna, si può definire la chiave sulla colonna con un vincolo di colonna al posto di un vincolo di tabella:

```
create table AUTORE
(NomeAutore  VARCHAR2(50) primary key,
Commenti     VARCHAR2(100));
```

In questo caso, la colonna `NomeAutore` è la chiave primaria, quindi Oracle creerà un nome per il vincolo **PRIMARY KEY**. Questa soluzione non è consigliata se si desidera imporre uno standard di denominazione comune per le chiavi, come discusso più avanti nel paragrafo “Assegnazione di nomi ai vincoli”.

## Designazione di tablespace di indici

I vincoli **UNIQUE** e **PRIMARY KEY** creano degli indici. A meno che non si specificino istruzioni diverse, Oracle collocherà questi indici nella tablespace predefinita (le tablespace sono descritte in modo dettagliato nel capitolo 20). Per specificare una tablespace diversa, occorre utilizzare la clausola `using index tablespace` del comando `create table`, come mostrato nel listato seguente:

```
create table AUTORE2
(NomeAutore  VARCHAR2(50),
Commenti     VARCHAR2(100),
constraint AUTORE_PK primary key (NomeAutore)
      using index tablespace UTENTI);
```

L’indice associato al vincolo di chiave primaria, `AUTORE_PK`, verrà inserito nella tablespace `UTENTI`. Per dettagli sull’uso delle tablespace, si rimanda al Capitolo 20.

**NOTA** In un’installazione predefinita, viene creata la tablespace `UTENTI`, che è anche la tablespace predefinita.

## La chiave esterna

Una *chiave esterna* è una combinazione di colonne con valori basati su quelli della chiave primaria di un’altra tabella. Un vincolo di chiave esterna, detto anche *vincolo di integrità referenziale*, specifica che i valori della chiave esterna corrispondono ai valori effettivi della chiave primaria

nell'altra tabella. Nella tabella BIBLIOTECA, per esempio, la colonna NomeCategoria fa riferimento ai valori della colonna NomeCategoria nella tabella CATEGORIA:

```
create table BIBLIOTECA
(Titolo      VARCHAR2(100) primary key,
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Valutazione  VARCHAR2(2),
constraint CATFK foreign key (NomeCategoria)
references CATEGORIA(NomeCategoria));
```

È possibile fare riferimento a una chiave primaria o unica, anche nella stessa tabella. Tuttavia, nella clausola `references` non si può fare riferimento a una tabella di un database remoto. Per specificare chiavi esterne con più colonne si può utilizzare la forma delle tabelle (impiegata nell'esempio precedente per creare una PRIMARY KEY sulla tabella STRANO) al posto della forma delle colonne.

A volte si desidera cancellare le righe dipendenti quando si elimina la riga da cui dipendono. Nel caso di BIBLIOTECA e CATEGORIA, se si elimina un NomeCategoria da CATEGORIA, si potrebbe voler rendere NULL i valori della colonna NomeCategoria corrispondente nella tabella BIBLIOTECA. Talvolta, si vuole cancellare l'intera riga. La clausola `on delete cascade` aggiunta alla clausola `references` comunica a Oracle di cancellare la riga dipendente quando viene cancellata la riga corrispondente nella tabella principale. Questa azione mantiene automaticamente l'integrità referenziale. Per ulteriori informazioni sulle clausole `on delete cascade` e `references`, si legga il paragrafo "Vincoli di integrità" nel Capitolo 42.

### **Il vincolo di controllo**

Molte colonne devono avere valori compresi in un determinato intervallo o che soddisfino particolari condizioni. Con un *vincolo di controllo* si può specificare un'espressione che dev'essere sempre vera per ogni riga della tabella. Per esempio, la tabella VALUTAZIONE memorizza le valutazioni valide; per limitare i valori disponibili oltre i limiti imposti dalla definizione della colonna si può ricorrere a un vincolo di controllo, come mostrato nel listato seguente:

```
create table VALUTAZIONE_CON_CONTROLLO
(Valutazione  VARCHAR2(2) CONTROLLO (Valutazione >= 9),
DescrizioneValutazione VARCHAR2(50));
```

Un vincolo di controllo a livello di colonna non può fare riferimento a valori in altre righe; e non può usare le pseudocolonne SysDate, UID, User, Userenv, Curval, Nextval, Level o RowNum. Si può utilizzare la forma del vincolo di tabella (invece di quella del vincolo di colonna) per fare riferimento a più colonne in un vincolo di controllo.

### **Assegnazione di nomi ai vincoli**

È possibile assegnare dei nomi ai vincoli. Se si ha uno schema efficiente di designazione per i nomi dei vincoli, sarà possibile identificare e gestire i vincoli stessi in modo migliore. Il nome di un vincolo dovrebbe identificare la tabella su cui agisce e il tipo di vincolo che rappresenta. Per esempio, la chiave primaria sulla tabella STRANO si potrebbe chiamare STRANO\_PK.

È possibile specificare un nome per un vincolo quando viene creato. Se non si specifica un nome per il vincolo, Oracle ne creerà uno. La maggior parte dei nomi di vincoli generati da Oracle è nella forma SYS\_C#####; per esempio SYS\_C000145. Dal momento che i nomi dei

vincoli generati dal sistema non offrono alcuna indicazione sulla tabella o sul vincolo, è opportuno assegnare sempre dei nomi ai vincoli.

Nell'esempio seguente, nel comando `create table` per la tabella STRANO viene creato un vincolo **PRIMARY KEY** e gli viene assegnato un nome. Si noti la clausola constraint:

```
create table STRANO (
Citta          VARCHAR2(13),
DataCampione   DATE,
Mezzogiorno    NUMBER(4,1),
Mezzanotte     NUMBER(4,1),
Precipitazione NUMBER,
constraint STRANO_PK PRIMARY KEY (Citta, DataCampione)
);
```

La clausola constraint del comando `create table` assegna un nome al vincolo (in questo caso `STRANO_PK`). In seguito, si potrà utilizzare questo nome quando si attiva o disattiva il vincolo.

## 18.2 Eliminazione di tabelle

Eliminare tabelle è molto semplice. Si utilizzano le parole `drop table`, seguite dal nome della tabella, come mostrato di seguito:

```
drop table STRANO;
```

```
Table dropped.
```

Le tabelle vengono eliminate quando non servono più. In Oracle, il comando `truncate` consente di rimuovere tutte le righe di una tabella e recuperare spazio per altre operazioni senza rimuovere la definizione della tabella dal database.

Anche il troncamento è molto semplice:

```
truncate table STRANO;
```

```
Table truncated.
```

Il troncamento non può essere annullato. Se esistono trigger che cancellano righe che dipendono da altre righe della tabella, questi non vengono eseguiti dal troncamento. Occorre essere *sicuri* di voler davvero effettuare il troncamento, prima di procedere con l'operazione.

## 18.3 Modifica di tabelle

Esistono tre modi per modificare le tabelle: aggiungendo una colonna a una tabella già esistente; modificando la definizione di una colonna; oppure eliminando una colonna. L'aggiunta di una colonna è semplice e simile alla creazione di una tabella. Si supponga di voler aggiungere due nuove colonne alla tabella STRANO: Condizione, che si ritiene debba essere NOT NULL, e Vento, per la velocità del vento. Il primo tentativo inizia creando la tabella e popolandola con alcuni record:

```

create table STRANO (
Citta      VARCHAR2(13),
DataCampione DATE,
Mezzogiorno NUMBER(4,1),
Mezzanotte NUMBER(4,1),
Precipitazione NUMBER);

insert into STRANO values
('PLEASANT LAKE','21-MAR-01',
39.99, -1.31, 3.6);

insert into STRANO values
('PLEASANT LAKE','22-JUN-01',
101.44, 86.2, 1.63);

insert into STRANO values
('PLEASANT LAKE','23-SEP-01',
92.85, 79.6, 1.00003);

insert into STRANO values
('PLEASANT LAKE','22-DEC-01',
-17.445, -10.4, 2.4);

```

Il primo tentativo di aggiungere le nuove colonne assume l'aspetto seguente:

```

alter table STRANO add (
Condizione  VARCHAR2(9) NOT NULL,
Vento       NUMBER(3)
);

alter table STRANO add (
*
ERROR at line 1: ORA-01758: table must be empty to add mandatory (NOT NULL)column

```

Si ottiene un messaggio di errore perché, logicamente, non è possibile aggiungere una colonna definita come NOT NULL, dato che è vuota quando si cerca di inserirla. Ogni riga nella tabella avrebbe una nuova colonna vuota definita come NOT NULL.

La clausola **add** del comando **alter table** funziona con una colonna NOT NULL se la tabella è vuota. Tuttavia, in genere non è pratico svuotare una tabella di tutte le sue righe solo per aggiungere una colonna NOT NULL. E non si possono utilizzare EXPORT e IMPORT se si aggiunge una colonna dopo l'esportazione ma prima dell'importazione.

L'alternativa consiste nel modificare la tabella aggiungendo inizialmente una colonna senza la restrizione NOT NULL:

```

alter table STRANO add (
Condizione  VARCHAR2(9),
Vento       NUMBER(3)
);

```

Table altered.

Quindi è necessario riempire la colonna con dei dati per ogni riga (dati validi oppure fintizi, finché non si ottengono quelli validi):

```
update STRANO set Condizione = 'SOLE';
```

Infine, si deve alterare nuovamente la tabella e modificare la definizione della colonna in NOT NULL:

```
alter table STRANO modify (
Condizione  VARCHAR2(9) NOT NULL,
Citta       VARCHAR2(17)
);
```

Table altered.

Si osservi che è stata modificata anche la colonna Citta per ampliarla a 17 caratteri (solo a scopo dimostrativo). Quando la tabella viene descritta, appare così:

```
describe STRANO
```

Nome	Null?	Type
CITTA	NOT NULL	VARCHAR2(17)
DATACAMPIONE	NOT NULL	DATE
MEZZOGIORNO		NUMBER(4,1)
MEZZANOTTE		NUMBER(4,1)
PRECIPITAZIOE		NUMBER
CONDIZIONE	NOT NULL	VARCHAR2(9)
VENTO		NUMBER(3)

La tabella contiene quanto segue:

```
select * from STRANO;
```

CITTA	DATACAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIP	CONDIZIONE	VENTO
PLEASANT LAKE	21-MAR-99	40	-1.3	3.6	SOLE	
PLEASANT LAKE	22-JUN-99	101.4	86.2	1.63	SOLE	
PLEASANT LAKE	23-SEP-99	92.9	79.6	1.00003	SOLE	
PLEASANT LAKE	22-DEC-99	-17.4	-10.4	2.4	SOLE	

Qui è possibile vedere gli effetti delle modifiche. Citta ora è ampia 17 caratteri anziché 13. Condizione è stata aggiunta alla tabella come NOT NULL e (temporaneamente) ha un valore "SOLE". È stata aggiunta anche Vento ed è NULL.

Per rendere annullabile una colonna NOT NULL, occorre utilizzare il comando alter table con la clausola NULL, come mostrato nel listato seguente:

```
alter table STRANO modify
(Condizione NULL);
```

## **Regole per aggiungere o modificare una colonna**

Di seguito sono elencate le regole per aggiungere una colonna a una tabella:

- Si può aggiungere una colonna in qualsiasi momento se non viene specificato NOT NULL.
- Si può aggiungere una colonna NOT NULL con tre passaggi:
  1. aggiungere la colonna senza specificare NOT NULL;
  2. riempire ogni riga di quella colonna con dei dati;
  3. modificare la colonna in NOT NULL.

**NOTA** Se si usa la clausola default quando si aggiunge una colonna, Oracle aggiornerà ogni riga con quel valore predefinito quando aggiunge la colonna; questi aggiornamenti avviano qualsiasi trigger AFTER UPDATE definito nella tabella (si faccia riferimento al Capitolo 28).

Di seguito sono elencate le regole per modificare una colonna:

- Si può aumentare la larghezza di una colonna di caratteri in qualsiasi momento.
- Si può aumentare il numero di cifre in una colonna NUMBER in qualsiasi momento.
- Si può aumentare o diminuire il numero di posizioni decimali in una colonna NUMBER in qualsiasi momento.

Inoltre, se una colonna è NULL per ogni riga della tabella si possono apportare le modifiche seguenti:

- Si può cambiare il tipo di dati della colonna.
- Si può diminuire la larghezza di una colonna di caratteri.
- Si può diminuire il numero di cifre in una colonna NUMBER.

Si può modificare il tipo di dati di una colonna solo se quest'ultima è vuota (NULL) in tutte le righe della tabella.

### Eccezioni per le modifiche alle colonne LONG-LOB

Esiste un'eccezione importante alle restrizioni per le modifiche dei tipi di dati. Oracle consente di convertire le colonne di tipo LONG in colonne di tipo LOB, anche se la colonna LONG contiene già dei dati. Il listato seguente illustra questa funzionalità:

```
create table TESTLONG
(Coll    NUMBER,
Longcol  LONG);

Table created.

insert into LONGTEST values (1,'Questo è un valore LONG');

1 row created.
```

```
alter table LONGTEST modify (Longcol CLOB);
```

```
Table altered.
```

```
desc LONGTEST
```

Nome	Null?	Type
COL1		NUMBER
LONGCOL		CLOB

```
select Longcol from TESTLONG;
```

```
LONGCOL
```

```
-----  
Questo è un valore LONG
```

## Eliminazione di una colonna

A partire da Oracle8i, esiste la possibilità di eliminare una colonna da una tabella. Eliminare una colonna è più complicato che aggiungerne o modificarne una, a causa del lavoro aggiuntivo che Oracle deve svolgere. Rimuovere la colonna dall'elenco di colonne nella tabella, in modo che non compaia quando si effettuano selezioni dalla tabella è semplice. È il recupero dello spazio effettivamente occupato dai valori della colonna il vero problema; infatti il database potrebbe impiegare molto tempo per effettuare questa operazione. Per questo motivo, è possibile eliminare una colonna immediatamente oppure contrassegnarla come “inutilizzata” in modo da eliminarla in seguito. Se la colonna viene eliminata immediatamente, questa azione potrebbe influire sulle prestazioni. Se la colonna invece viene contrassegnata come inutilizzata, non ci sarà nessun impatto sulle prestazioni. La colonna potrà essere eliminata in un secondo tempo, quando il database non è così carico di lavoro.

Per eliminare una colonna, si utilizzi la clausola `set unused` oppure la clausola `drop` del comando `alter table`. Non è possibile eliminare una pseudocolonna, una colonna di una tabella annidata oppure la colonna di una chiave di partizione.

Nell'esempio seguente, la colonna Vento viene eliminata dalla tabella STRANO:

```
alter table STRANO drop column Vento;
```

In alternativa, si può contrassegnare la colonna Vento come inutilizzata:

```
alter table STRANO set unused column Vento;
```

Se si contrassegna una colonna come “inutilizzata”, lo spazio precedentemente utilizzato da questa colonna non viene rilasciato fino a quando non si eliminano le colonne inutilizzate:

```
alter table STRANO drop unused columns;
```

È possibile eseguire una query su `USER_UNUSED_COL_TABS`, `DBA_UNUSED_COL_TABS` e `ALL_UNUSED_COL_TABS` per visualizzare tutte le tabelle con le colonne contrassegnate come inutilizzate.

**NOTA** *Una volta contrassegnata come “inutilizzata” una colonna non sarà più accessibile.*

Si possono eliminare più colonne con un solo comando, come mostrato nel listato seguente:

```
alter table STRANO drop (Condizione, Vento);
```

**NOTA** *Quando si eliminano più colonne, non si dovrebbe utilizzare la parola chiave `column` del comando `alter table`, perché causa un errore di sintassi. I nomi delle colonne devono essere racchiusi tra parentesi, come mostrato nel listato precedente.*

Se le colonne eliminate fanno parte di chiavi primarie o di vincoli unici, si dovrà utilizzare anche la clausola `cascade constraints` nel comando `alter table`. Se si elimina una colonna che appartiene a una chiave primaria, Oracle eliminerà la colonna e l'indice della chiave primaria.

## 18.4 Creazione di una vista

Dato che le tecniche per creare una vista sono state già descritte nei capitoli precedenti, non vengono trattate di nuovo. Questo paragrafo tuttavia introduce alcuni concetti relativi alle viste che si dimostreranno utili.

Se una vista viene creata partendo da un'unica tabella di base, è possibile inserire, aggiornare o cancellare righe nella vista stessa. In questo modo vengono effettivamente inserite, aggiornate o cancellate righe nella tabella di base. L'esecuzione di queste operazioni è soggetta ad alcune restrizioni, abbastanza ovvie:

- Non è possibile effettuare inserimenti se la tabella di base ha una colonna NOT NULL che non compare nella vista.
- Non è possibile effettuare inserimenti o aggiornamenti se qualcuna delle colonne della vista, cui si fa riferimento in `insert` o `update`, contiene funzioni o calcoli.
- Non è possibile effettuare inserimenti, aggiornamenti o cancellazioni se la vista contiene `group by`, `distinct` o un riferimento alla pseudocolonna `RowNum`.

È possibile eseguire inserimenti in una vista basata su più tabelle se Oracle è in grado di stabilire le righe corrette da inserire. In una vista a più tabelle, Oracle stabilisce per quali tabelle viene *mantenuta la chiave*. Se una vista contiene un numero sufficiente di colonne, tratte da una tabella, per identificare la chiave primaria della tabella, la chiave viene conservata e Oracle può inserire righe nella tabella tramite la vista.

## Stabilità di una vista

Si ricordi che i risultati di una query su una vista sono costruiti istantaneamente da una tabella (o da più tabelle) quando si esegue la query. Fino a quel momento la vista non ha dati propri, come li ha una tabella. È semplicemente una descrizione (un'istruzione SQL) delle informazioni da prelevare da altre tabelle e di come organizzarle. Di conseguenza, se una tabella viene eliminata, la vista non ha più alcuna validità. Se si cerca di eseguire una query su una vista la cui tabella di base è stata eliminata, si ottiene un messaggio di errore relativo alla vista.

**NOTA** *L'unica eccezione a questa regola riguarda l'uso di viste materializzate, introdotte in Oracle8i. In realtà, una vista materializzata è una tabella che memorizza i dati per i quali normalmente verrebbe eseguita una query tramite una vista. Le viste materializzate sono descritte in dettaglio nel Capitolo 23.*

Nella sequenza riportata sotto si crea una vista su una tabella già esistente, si rimuove la tabella e poi si effettua una query sulla vista.

Innanzitutto, viene creata la vista:

```
create view PIOGGIA as
select Citta, Precipitazione
  from STRANO;
```

View created.

La tabella di base viene eliminata:

```
drop table STRANO;
```

Table dropped.

Infine, viene effettuata la query sulla vista:

```
select * from PIOGGIA;
*
ERROR at line 1: ORA-04063: view "PRATICA.PIOGGIA" has errors
```

Analogamente, viene creata una vista utilizzando l'asterisco:

```
create view PIOGGIA as
select * from STRANO;
```

View created.

e poi viene modificata la tabella di base:

```
alter table STRANO add (
Attenzione      VARCHAR2(20)
);
```

Table altered.

Nonostante la modifica alla tabella di base della vista, quest'ultima è ancora valida, anche se le colonne nella vista sono soltanto quelle che esistevano in CREATE VIEW.

Dopo aver modificato la tabella, è necessario sostituire la vista. Per creare nuovamente una vista conservando tutti i privilegi che le sono stati concessi, occorre utilizzare il comando `create or replace view`, come mostrato nel listato seguente. Questo comando sostituisce il testo di una vista già esistente con quello della nuova vista, senza influire sulle vecchie concessioni.

```
create or replace view PIOGGIA as
select * from STRANO;
```

## **order by nelle viste**

A partire da Oracle8i, si ha la possibilità di utilizzare `order by` in un'istruzione `create view`, come mostrato nel listato seguente basato sulla tabella STRANO ricostruita. Si osservi che le query effettuate su questa vista subiscono un calo delle prestazioni a causa del lavoro di ordinamento aggiuntivo che devono svolgere.

```
create view STRANO_ORDINATA
  as select * from STRANO
  order by Citta, DataCampione;
```

View created.

```
select * from STRANO_ORDINATA
```

CITTA	DATA CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
PLEASANT LAKE	21-MAR-01	4.4	-18.5	3.6
PLEASANT LAKE	22-JUN-01	38.5	30.1	1.63
PLEASANT LAKE	23-SEP-01	33.8	26.4	1.00003
PLEASANT LAKE	22-DEC-01	-27.4	-23.5	2.4

## Creazione di una vista di sola lettura

È possibile utilizzare la clausola `with read only` del comando `create view` per evitare che gli utenti manipolino i dati attraverso la vista. Si consideri la vista PIOGGIA creata nel paragrafo precedente:

```
create or replace view PIOGGIA as
select * from STRANO;
```

Se un utente è in grado di cancellare record dalla tabella STRANO, potrebbe cancellare i record di STRANO attraverso la vista PIOGGIA:

```
delete from PIOGGIA;
```

L'utente potrebbe anche inserire o aggiornare record nella tabella STRANO eseguendo queste operazioni sulla vista PIOGGIA. Se la vista è basata sul join di più tabelle, la capacità dell'utente di aggiornare i record della vista è limitata; le tabelle di base di una vista non possono essere aggiornate, a meno che nell'aggiornamento non sia coinvolta una sola tabella e la chiave primaria completa della tabella aggiornata non sia inclusa nelle colonne della vista.

Per impedire che le tabelle di base vengano modificate attraverso una vista, si può utilizzare la clausola `with read only` del comando `create view`. Se si usa questa clausola quando si crea una vista, gli utenti potranno selezionare *soltanto* record dalla vista. Gli utenti non potranno manipolare i record ottenuti dalla vista, anche se questa è basata su una sola tabella:

```
create or replace view PIOGGIA_SOLA_LETTURA as
select * from STRANO
with read only;
```

Si può anche utilizzare un trigger `INSTEAD OF` per gestire i comandi di manipolazione dei dati eseguiti sulla vista. Per ulteriori dettagli sui trigger `INSTEAD OF`, si rimanda al Capitolo 30.

## 18.5 Creazione di una tabella a partire da un'altra

La vista PIOGGIA creata in precedenza a partire dalla tabella STRANO sarebbe anche potuta essere una tabella. Oracle consente di creare al momento una nuova tabella, basata su un'istruzione `select` in una tabella già esistente:

```
create table PIOGGIA_TABELLA as
select Citta, Precipitazione
from STRANO;
```

Table created.

**NOTA** *Il comando `create table...as select...` non funziona se una delle colonne selezionate utilizza il tipo di dati LONG.*

Quando la nuova tabella viene descritta, si scopre che ha “ereditato” le definizioni delle colonne dalla tabella STRANO. Una tabella creata in questo modo può includere tutte le colonne, utilizzando un asterisco, o un sottoinsieme delle colonne di un'altra tabella. Può includere anche delle colonne “inventate”, che sono il prodotto di funzioni o la combinazione di altre

colonne, proprio come per una vista. Le definizioni delle colonne di caratteri si adattano alla dimensione necessaria a contenere i dati nelle colonne inventate. Le colonne NUMBER che avevano una precisione specificata nella tabella di origine, ma che vengono sottoposte a calcoli per la creazione di una nuova colonna, nella nuova tabella sono semplicemente colonne NUMBER con una precisione non specificata. Quando la tabella PIOGGIA viene descritta, mostra le definizioni delle colonne:

```
describe PIOGGIA_TABELLA
```

Nome	Null?	Type
CITTA	NOT NULL	VARCHAR2(13)
PRECIPITAZIONE		NUMBER

Quando viene effettuata una query su PIOGGIA\_TABELLA, si può vedere che questa contiene proprio le colonne e i dati selezionati dalla tabella STRANO, così come erano quando PIOGGIA\_TABELLA è stata creata:

```
select * from PIOGGIA_TABELLA;
```

CITTA	PRECIPITAZIONE
PLEASANT LAKE	3.6
PLEASANT LAKE	1.63
PLEASANT LAKE	1.00003
PLEASANT LAKE	2.4

Si può utilizzare questa tecnica anche per creare una tabella con definizioni di colonne come quelle della tabella di origine, ma senza righe, costruendo una clausola where che non seleziona alcuna riga dalla vecchia tabella:

```
create table PIOGGIA_VUOTA as
select Citta, Precipitazione
  from STRANO
 where 1=2;
```

Table created.

Una query su questa tabella mostra che essa non contiene nulla:

```
select * from PIOGGIA_VUOTA;
```

no rows selected.

Si può creare una tabella basata su una query senza generare *voci redo log* (registrazioni cronologiche delle azioni del database utilizzate durante i ripristini). Per evitare la creazione di queste voci si utilizza la parola chiave nologging nel comando create table. Quando si evitano in questo modo le voci redo log, le prestazioni del comando create table migliorano, perché c'è meno lavoro da svolgere; più è grande la tabella, maggiore è l'impatto. Tuttavia, poiché la creazione della nuova tabella non viene scritta nei *file redo log* (che registrano le voci redo log), la tabella non viene ricreata se, dopo un guasto del database, si utilizzano questi file redo log per ripristinarlo. Di conseguenza, conviene effettuare un backup del database subito dopo l'utilizzo dell'opzione nologging, se si desidera avere la possibilità di recuperare la nuova tabella.

L'esempio seguente mostra come utilizzare la parola chiave `unrecoverable` durante la creazione di tabelle basate su query. Per default, la creazione di tabelle basate su query genera voci redo log.

```
create table PIOGGIA_NOLOG
nologging
as
select * from STRANO;
```

Table created.

Si osservi la possibilità di specificare `nologging` a livello della partizione e a livello del LOB. L'utilizzo di `nologging` migliora le prestazioni delle operazioni che creano i dati iniziali nella tabella.

## 18.6 Creazione di una tabella di solo indice

Una *tabella di solo indice* mantiene i dati ordinati in base ai valori della colonna primaria. Le tabelle di solo indice memorizzano i dati come se l'intera tabella fosse registrata in un indice. Gli indici sono descritti più dettagliatamente nel Capitolo 20; in Oracle gli indici servono per due scopi principali:

- **Per imporre l'unicità** Quando viene creato un vincolo `PRIMARY KEY` o `UNIQUE`, Oracle produce un indice per imporre l'unicità delle colonne indicizzate.
- **Per migliorare le prestazioni** Quando una query può utilizzare un indice, le sue prestazioni migliorano considerevolmente. Per informazioni dettagliate sulle condizioni in cui una query può utilizzare un indice, si rimanda al Capitolo 38.

Una tabella di solo indice consente di memorizzare i dati dell'intera tabella in un indice. Un indice normale memorizza nell'indice solo le colonne indicizzate; una tabella di solo indice memorizza nell'indice tutte le colonne della tabella.

Per creare una tabella di solo indice, si deve utilizzare la clausola `organization index` del comando `create table`, come mostrato nell'esempio seguente:

```
create table STRANO_IOT (
Citta      VARCHAR2(13),
DataCampione DATE,
Mezzogiorno NUMBER(4,1),
Mezzanotte NUMBER(4,1),
Precipitazione NUMBER,
constraint STRANO_IOT_PK PRIMARY KEY (Citta, DataCampione))
organization index;
```

Per creare `STRANO` come tabella di solo indice, è necessario stabilire per essa un vincolo `PRIMARY KEY`.

Una tabella di solo indice è appropriata se si accede sempre ai dati di `STRANO` attraverso le colonne `Città` e `DataCampione` (nelle clausole `where` delle query). Per ridurre al minimo la quantità di lavoro necessario per la gestione attiva dell'indice, conviene utilizzare tabelle di solo indice soltanto se i dati sono molto statici. Se questi cambiano frequentemente o se si ha la necessità di indicizzare altre colonne della tabella, conviene impiegare una tabella normale, con gli indici appropriati.

In linea di massima, le tabelle di solo indice sono le più efficienti quando la chiave primaria costituisce una grossa parte delle colonne della tabella. Se la tabella contiene molte colonne ad accesso frequente che non fanno parte della chiave primaria, la tabella di solo indice dovrà accedere ripetutamente alla propria area di overflow. Nonostante questo aspetto negativo, si potrebbe decidere di usare le tabelle di solo indice per sfruttare una caratteristica fondamentale che non è disponibile con le tabelle standard: la possibilità di utilizzare l'opzione move online del comando alter table. Questa opzione può essere utilizzata per spostare una tabella da una tablespace in un'altra mentre vi accedono i comandi insert, update e delete. Non è possibile invece usare l'opzione move online per le tabelle partizionate di solo indice. Il partizionamento viene descritto nel prossimo paragrafo.

## 18.7 Uso di tabelle partizionate

A partire da Oracle8, è possibile suddividere le righe di una singola tabella in più parti. Questa particolare suddivisione dei dati di una tabella prende il nome di *partizionamento* della tabella; la tabella che viene partizionata è detta *tabella partizionata* e le parti sono dette *partizioni*.

Il partizionamento è utile per tabelle molto grandi. Suddividendo le righe di una tabella di grandi dimensioni in varie partizioni più piccole, si realizzano diversi obiettivi importanti:

- Le prestazioni delle query sulle tabelle possono migliorare, in quanto per risolvere una query, Oracle dovrà cercare solo in una partizione (una parte della tabella) invece che in un'intera tabella.
- La tabella può essere più semplice da gestire. Dal momento che i dati delle tabelle partizionate vengono memorizzati in più parti, può essere più semplice caricare e cancellare dati nelle partizioni invece che nella tabella grande.
- Le operazioni di backup e recupero possono essere svolte in modo migliore. Dato che le partizioni sono più piccole della tabella partizionata, si possono avere più opzioni per effettuare il backup e il recupero delle partizioni che per una sola grande tabella.

L'ottimizzatore di Oracle comprende che la tabella è stata partizionata; come mostrato più avanti nel capitolo, è anche possibile specificare la partizione da utilizzare nella clausola from delle query.

### Creazione di una tabella partizionata

Per creare una tabella partizionata, è necessario specificare in che modo devono essere configurate le partizioni dei dati della tabella nel comando create table. Solitamente, le tabelle vengono partizionate in base a intervalli di valori.

Si consideri la tabella BIBLIOTECA:

```
create table BIBLIOTECA
(Titolo      VARCHAR2(100) primary key,
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Valutazione  VARCHAR2(2),
Constraint CATFK foreign key (NomeCategoria)
references CATEGORIA(NomeCategoria));
```

Se si intende memorizzare un gran numero di record nella tabella BIBLIOTECA, è preferibile dividere le righe in più partizioni. Per partizionare i record della tabella, occorre utilizzare la

clausola `partition by range` del comando `create table`, come mostrato più avanti. Gli intervalli stabiliscono i valori memorizzati in ogni partizione.

```
create table BIBLIOTECA_INTERVALLO_PART
(Titolo      VARCHAR2(100) primary key,
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Valutazione  VARCHAR2(2),
constraint CATFK2 foreign key (NomeCategoria)
references CATEGORIA(NomeCategoria)
)
partition by range (NomeCategoria)
(partition PART1    values less than ('B')
  tablespace PART1_TS,
 partition PART2    values less than (MAXVALUE)
  tablespace PART2_TS);
/
```

La tabella BIBLIOTECA viene partizionata in base ai valori della colonna NomeCategoria:

```
partition by range (NomeCategoria)
```

Per qualsiasi valore di Categoria minore di ‘B’ (le categorie ADULTI), il record viene memorizzato nella partizione chiamata PART1. Questa partizione viene memorizzata nella tablespace PART1\_TS (per maggiori dettagli sulle tablespace si rimanda al Capitolo 20). Qualsiasi altra categoria viene memorizzata nella partizione PART2. Si noti che nella definizione di quest’ultima partizione la clausola dell’intervallo è la seguente:

```
partition PART2    values less than (MAXVALUE)
```

Non è necessario specificare un valore massimo per l’ultima partizione; la parola chiave MAXVALUE comunica a Oracle di utilizzare la partizione per memorizzare qualsiasi dato non sia stato memorizzato nelle partizioni precedenti.

Si possono creare più partizioni, ciascuna con un proprio valore superiore. Per ogni partizione, è necessario specificare solamente il valore massimo dell’intervallo. Il valore minimo viene determinato implicitamente da Oracle.

Le partizioni a intervalli erano le uniche consentite in Oracle8. Oracle8i ha introdotto le *partizioni hash*. Una partizione hash determina la collocazione fisica dei dati eseguendo una funzione hash sui valori della chiave di partizione. Nel *partizionamento a intervalli*, in genere i valori consecutivi della chiave di partizione vengono memorizzati nella stessa partizione. Nell’*hash partitioning*, i valori consecutivi della chiave di partizione non sono memorizzati necessariamente nella stessa partizione. L’hash partitioning distribuisce un insieme di record tra un gruppo di partizioni più ampio rispetto al partizionamento a intervalli, riducendo le probabilità di conflitti di I/O.

Per creare una partizione hash, al posto della clausola `partition by range` occorre utilizzare la clausola `partition by hash`, come mostrato nel listato seguente:

```
create table BIBLIOTECA_HASH_PART
(Titolo      VARCHAR2(100) primary key,
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Valutazione  VARCHAR2(2),
constraint CATFK_HASH foreign key (NomeCategoria)
```

```

references CATEGORIA(Nomecategoria)
)
partition by hash (NomeCategoria)
partitions 10;

```

Si può denominare ogni partizione e specificare la tablespace, proprio come si farebbe per il partizionamento a intervalli, come mostrato di seguito:

```

create table BIBLIOTECA_HASH_PART
(Titolo      VARCHAR2(100) primary key,
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Valutazione  VARCHAR2(2),
constraint CATFK_HASH foreign key (NomeCategoria)
references CATEGORIA(NomeCategoria)
)
partition by hash (NomeCategoria)
partitions 2
store in (PART1_TS, PART2_TS);

```

Dopo la linea `partition by hash (NomeCategoria)`, ci sono due opzioni per il formato:

- Come mostrato nel listato precedente, è possibile specificare il numero di partizioni e di tablespace da utilizzare:

```

partitions 2
store in (PART1_TS, PART2_TS);

```

Questo metodo produce partizioni con nomi creati dal sistema e con un formato `SYS_Pnnn`. Il numero delle tablespace specificato nella clausola `store in` non deve obbligatoriamente essere uguale al numero delle partizioni. Se si specificano più partizioni che tablespace, le partizioni verranno assegnate alle tablespace seguendo uno schema circolare.

- Si possono specificare anche partizioni con nome:

```

partition by hash (Nomecategoria)
(partition PART1 tablespace PART1_TS,
 partition PART2 tablespace PART2_TS);

```

In questo modo, a ogni partizione vengono assegnati un nome e una tablespace, con la possibilità di usare una clausola di memorizzazione aggiuntiva `lob` o `varray` (si consultino i Capitoli 31 e 32). Questo metodo offre un maggior controllo sulle posizioni delle partizioni, oltre al grande vantaggio di poter specificare nomi significativi per le partizioni.

### **Creazione di partizioni secondarie**

A partire da Oracle8i, esiste la possibilità di creare *partizioni secondarie*, ovvero partizioni di partizioni. Le partizioni secondarie possono essere utilizzate per combinare i due tipi di partizioni: le partizioni a intervalli e le partizioni hash. Le partizioni hash possono essere usate insieme a quelle a intervalli, creando così partizioni hash delle partizioni a intervalli. Per le tabelle di grandi dimensioni, questo partizionamento composito può rappresentare un modo efficace per separare i dati in divisioni gestibili e regolabili.

L'esempio seguente crea partizioni a intervalli per la tabella BIBLIOTECA in base ai valori della colonna Titolo, e partizioni hash delle partizioni di Titolo in base ai valori di NomeCategoria:

```

create table BIBLIOTECA_INTERVALLO_HASH_PART
(Titolo      VARCHAR2(100) primary key,
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Valutazione  VARCHAR2(2),
constraint CATFK3 foreign key (NomeCategoria)
references CATEGORIA(NomeCategoria)
)
partition by range (Titolo)
subpartition by hash (NomeCategoria)
subpartitions 6
(partition PART1    values less than ('M')
  tablespace PART1_TS,
partition PART2    values less than (MAXVALUE)
  tablespace PART2_TS);

```

La tabella BIBLIOTECA avrà partizioni a intervalli in due partizioni, usando gli intervalli dei valori di Titolo specificati per le partizioni nominate. Per ciascuna di queste partizioni verrà creata una partizione hash nella colonna NomeCategoria.

## Partizionamento a elenco

A partire da Oracle9i è possibile utilizzare partizioni a elenco al posto del partizionamento a intervalli e di quello hash. Nel partizionamento a elenco, si indicano a Oracle tutti i valori possibili, e si stabiliscono le partizioni in cui dovrebbero essere inserite le righe corrispondenti. L'esempio seguente mostra le voci della tabella BIBLIOTECA partizionate a elenco in base ai valori di NomeCategoria:

```

create table BIBLIOTECA_LIST_PART
(Titolo      VARCHAR2(100) primary key,
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Valutazione  VARCHAR2(2),
constraint CATFK4 foreign key (NomeCategoria)
references CATEGORIA(NomeCategoria)
)
partition by list (NomeCategoria)
(partition PART1    values ('NARRAADULTI', 'SAGGIADULTI', 'CONSADULTI')
  tablespace PART1_TS,
partition PART2    values
  ('NARRRAGAZZI', 'SAGGIRAGAZZI', 'ILLUSRAGAZZI')
  tablespace PART1_TS);

```

**NOTA** In Oracle9i Release 9.0.1, il tentativo di inserire una riga con un valore che non esiste nelle definizioni delle partizioni a elenco, in questo caso un nome di categoria differente, provocherà un errore. Questa limitazione verrà eliminata in una prossima versione di Oracle9i.

## Indicizzazione delle partizioni

Quando si crea una tabella partizionata, si dovrebbe creare un indice su di essa. L'indice può essere partizionato in base agli stessi intervalli che sono stati impiegati per partizionare la tabella.

Ia. Nel listato seguente, viene mostrato il comando `create index` per la tabella `BIBLIOTECA_LIST_PART` (partizionata a elenco). Le partizioni dell'indice sono inserite nelle tablespace `PART1_NDX_TS` e `PART2_NDX_TS`.

```
create index BIBLIOTECA_ELENCO_CATEGORIA
on BIBLIOTECA_LIST_PART(NomeCategoria)
local
(partition PART1
  tablespace PART1_NDX_TS,
partition PART2
  tablespace PART2_NDX_TS);
```

Si noti la parola chiave `local`. In questo comando `create index` non viene specificato alcun intervallo, mentre invece la parola chiave `local` comunica a Oracle di creare un indice distinto per ogni partizione della tabella `BIBLIOTECA_LIST_PART`. In `BIBLIOTECA_LIST_PART` sono state create due partizioni. Pertanto, questo indice crea due indici separati, uno per ogni partizione. Dato che esiste un indice per partizione, gli indici sono “locali” rispetto alle partizioni.

Si possono creare anche indici “globali”. Un indice globale può contenere valori di più partizioni.

```
create index BIBLIOTECA_ELENCO_CATEGORIA_G
on BIBLIOTECA_LIST_PART(Editore)
global;
)
```

La clausola `global` in questo comando `create index` consente di creare un indice non partizionato (come mostrato in questo esempio) oppure di specificare intervalli per i valori degli indici diversi dagli intervalli per le partizioni della tabella. Gli indici locali possono essere più semplici da gestire di quelli globali; tuttavia, gli indici globali possono svolgere controlli di unicità più veloci di quelli locali (partizionati).

**NOTA** *Non è possibile creare indici globali per le partizioni o per le partizioni secondarie hash.*

## Gestione delle tabelle partizionate

Si può utilizzare il comando `alter table` per aggiungere, rimuovere, permutare, spostare, modificare, rinominare, dividere e troncare le partizioni. Queste opzioni del comando `alter table` consentono di modificare la struttura delle partizioni già esistenti, come potrebbe risultare necessario dopo che una tabella partizionata è stata intensamente utilizzata. Per esempio, la distribuzione dei valori di `NomeCategoria` all'interno di una tabella partizionata può essere cambiata, oppure il valore massimo può essere aumentato. Per ulteriori informazioni su queste opzioni, si può consultare la voce “`ALTER TABLE`” nel Capitolo 42.

## Realizzazione di query direttamente dalle partizioni

Se si conosce la partizione dalla quale possono essere recuperati i dati, è possibile specificarne il nome nella clausola `from` della query. Per esempio, si supponga di voler eseguire una query dei record per i libri contenuti nelle categorie `ADULTI`. L'ottimizzatore dovrebbe essere in grado di utilizzare le definizioni delle partizioni per stabilire che solo la partizione `PART1` può contenere

i dati che risolvono questa query; se lo si desidera, si può indicare esplicitamente a Oracle di utilizzare PART1 nella query:

```
insert into BIBLIOTECA_LIST_PART select * from BIBLIOTECA;
select COUNT(*) from BIBLIOTECA_LIST_PART partition (part1);
COUNT(*)
-----
22
```

Questo esempio nomina esplicitamente la partizione in cui Oracle dovrà cercare i record della categoria ADULTI. Se questa partizione viene modificata (per esempio, se il suo intervallo di valori viene alterato), PART1 potrebbe non essere più la partizione che contiene questi record. Pertanto, occorre prestare molta attenzione quando si usa questa sintassi.

Generalmente, questa sintassi non è necessaria, perché Oracle memorizza le definizioni degli intervalli per le varie partizioni. Quando si effettua una query dalla tabella partizionata, Oracle stabilisce quali partizioni dovrebbero essere utilizzate per risolvere questa query. Questo processo può fare in modo che la ricerca della query venga effettuata solo su un piccolo numero di righe, migliorando le prestazioni della query stessa. Inoltre, le partizioni possono essere memorizzate in tablespace diverse (e quindi su dischi distinti), contribuendo a ridurre la probabilità di conflitto per l'I/O del disco durante lo svolgimento della query.

Durante un inserimento nella tabella partizionata, Oracle utilizza le definizioni delle partizioni per stabilire in quale partizione dovrebbe essere inserito il record. Quindi, si può utilizzare una tabella partizionata come se fosse una tabella singola e affidarsi a Oracle per la gestione della separazione interna dei dati.

## 18.8 Ridefinizione in linea delle tabelle

Come descritto in precedenza in questo capitolo, è possibile eseguire diversi comandi per la ridefinizione degli oggetti in linea, compreso lo spostamento delle tabelle di solo indice, la sostituzione delle viste e l'eliminazione delle colonne. A partire da Oracle9i esiste la possibilità di ridefinire le tabelle in linea; è addirittura possibile creare partizioni di una tabella non partizionata mentre è accessibile per gli utenti. Si può utilizzare questa caratteristica per spostare le tabelle tra le tablespace nelle applicazioni di elaborazione di transazioni in linea.

Per ottenere questo risultato mentre il database è accessibile, occorre usare il package DBMS\_REDEFINITION fornito con Oracle9i. Questo processo è soggetto ad alcune restrizioni significative (tratte dalla *Oracle9i Administrator's Guide*).

- Per essere candidate alla ridefinizione in linea, le tabelle devono possedere delle chiavi primarie.
- La tabella da ridefinire e l'ultima tabella ridefinita devono avere la stessa colonna di chiave primaria.
- Le tabelle in cui sono definiti viste materializzate e log delle viste materializzate non possono essere ridefinite in linea.
- Le tabelle contenitore di viste materializzate e le tabelle di Advanced Queuing non possono essere ridefinite in linea.
- La tabella di overflow di una tabella di solo indice non può essere ridefinita in linea.

- Le tabelle con tipi definiti dall'utente (oggetti, REF, collezioni e tabelle con vari tipi) non possono essere ridefinite in linea.
- Le tabelle con colonne FILE non possono essere ridefinite in linea.
- Le tabelle con colonne LONG non possono essere ridefinite in linea, mentre le tabelle con colonne LOB sono accettabili.
- La tabella da ridefinire non può fare parte di un cluster.
- Le tabelle nello schema SYS e SYSTEM non possono essere ridefinite in linea.
- Le tabelle temporanee non possono essere ridefinite in linea.
- Non esiste un supporto orizzontale per la creazione di subset (in altre parole, nella tabella di origine non è possibile specificare alcuna clausola where).
- Quando si associano le colonne della tabella temporanea a quelle della tabella originale si possono usare solamente espressioni deterministiche semplici. Per esempio, le subquery non sono consentite.
- Se vengono aggiunte nuove colonne (che non sono istanziate con dati esistenti per la tabella originale) come parte della ridefinizione, non dovranno essere dichiarate come NOT NULL, fino a quando la ridefinizione non è terminata.
- Non ci possono essere vincoli referenziali tra la tabella ridefinita e quella temporanea.

Se le tabelle soddisfano questi criteri, è possibile avviare il processo di ridefinizione, costituito da sei passaggi:

1. Verificare che la tabella possa essere ricostruita in linea.
2. Creare la tabella temporanea con i suoi indici, concessioni, vincoli e trigger.
3. Avviare la ridefinizione.
4. Facoltativamente, interrompere il processo o sincronizzare le tabelle di origine/destinazione.
5. Terminare il processo di ridefinizione.
6. Eliminare la tabella temporanea.

I passaggi seguenti mostrano la ridefinizione della tabella AUTORE durante l'esecuzione del partizionamento mentre è accessibile agli utenti.

## **1. Verificare che la tabella possa essere ricostruita in linea**

Per verificare che la tabella possa essere ricostruita in linea, occorre eseguire la procedura CAN\_REDEF\_TABLE, fornendo come input il nome del proprietario dello schema e il nome della tabella:

```
execute DBMS_REDEFINITION.CAN_REDEF_TABLE('PRATICA','AUTORE');
```

Se viene segnalata una serie di errori, il primo errore dell'elenco indicherà la natura del problema; successivamente, gli errori relativi al package DBMS\_REDEFINITION potranno essere ignorati.

## **2. Creare la tabella temporanea**

Come si desidera che appaia la tabella AUTORE nella nuova tablespace? Si crei una tabella temporanea che verrà utilizzata durante il processo di ridefinizione. Per semplificare le operazioni, è opportuno mantenere i nomi delle colonne e l'ordine uguali a quelli della tabella AUTORE di origine.

```

create table AUTORE_TEMPORANEA
(NomeAutore  VARCHAR2(50) primary key,
Commenti      VARCHAR2(100)
)
partition by range (NomeAutore)
(partition PART1 values less than ('M'),
partition PART2 values less than (MAXVALUE)
);

```

In questo esempio, la tabella AUTORE\_TEMPORANEA fungerà da tabella temporanea durante la ridefinizione. Terminato il processo, AUTORE\_TEMPORANEA sostituirà la vecchia tabella AUTORE, e gli unici indici, vincoli, trigger e concessioni disponibili per AUTORE saranno quelli creati per AUTORE\_TEMPORANEA prima dell'ultimo passaggio (ossia, il passaggio 5 in questa descrizione). Facoltativamente, la creazione di indici, vincoli, trigger e concessioni può anche avvenire al termine del processo di ridefinizione, tuttavia la loro creazione eviterà la necessità di lock per le tabelle attive.

### **3. Avviare la ridefinizione**

La procedura START\_REDEF\_TABLE avvia la ridefinizione. La tabella AUTORE\_TEMPORANEA verrà popolata con i dati su cui è stato eseguito il commit nella tabella AUTORE, quindi i requisiti per i segmenti di undo e di tempo (dimensione della transazione) per questo passaggio dipendono dalla dimensione della tabella AUTORE. Come parametri si devono passare il nome del proprietario dello schema, il nome della vecchia tabella e il nome della tabella temporanea:

```

execute DBMS_REDEFINITION.START_REDEF_TABLE -
('PRATICA','AUTORE','AUTORE_TEMPORANEA');

```

Se AUTORE\_TEMPORANEA avesse un elenco di colonne diverso da quello di AUTORE, questo elenco di colonne verrebbe passato come quarto parametro alla procedura START\_REDEF\_TABLE. Completata questa procedura, si potrà eseguire una query su AUTORE\_TEMPORANEA per verificarne i dati, Oracle l'avrà popolata con i dati presi da AUTORE.

### **4(a). FACOLTATIVO – Interrompere il processo**

Se a questo punto diventa necessario interrompere il processo di ridefinizione, occorre utilizzare la procedura ABORT\_REDEF\_TABLE, inserendo come parametri di input il nome del proprietario dello schema, la tabella di origine e la tabella temporanea.

```

execute DBMS_REDEFINITION.ABORT_REDEF_TABLE -
('PRATICA','AUTORE','AUTORE_TEMPORANEA');

```

Dopo aver interrotto la ridefinizione, sarebbe opportuno troncare la tabella temporanea (AUTORE\_TEMPORANEA) anche per reclamare l'allocazione del suo spazio.

### **4(b). FACOLTATIVO – Sincronizzare le tabelle**

Durante la fase conclusiva del processo di ridefinizione, Oracle sincronizzerà la data tra la tabella di origine e quella temporanea. Per abbreviare il tempo richiesto da questa parte del processo, è possibile sincronizzare le tabelle prima dell'ultimo passaggio. Con questo passaggio facoltativo si può istanziare l'ultimissima data di produzione nella tabella temporanea, riducendo al minimo l'impatto sugli utenti in linea.

Per sincronizzare la tabella di origine e quella temporanea, occorre usare la procedura `SYNC_INTERIM_TABLE`, inserendo come parametri di input il nome del proprietario dello schema, la tabella di origine e quella temporanea.

```
execute DBMS_REDEFINITION.SYNC_INTERIM_TABLE -
  ('PRATICA','AUTORE','AUTORE_TEMPORANEA');
```

## 5. Terminare il processo di ridefinizione

Per completare il processo di ridefinizione, si esegua la procedura `FINISH_REDEF_TABLE`, come mostrato negli esempi seguenti. I parametri di input sono i nomi del proprietario, della tabella di origine e della tabella temporanea.

```
execute DBMS_REDEFINITION.FINISH_REDEF_TABLE -
  ('PRATICA','AUTORE','AUTORE_TEMPORANEA');
```

In questo passaggio, la tabella AUTORE acquisisce le caratteristiche della tabella AUTORE\_TEMPORANEA, diventando quindi una tabella partizionata, e assume il valore ID dell'oggetto interno di AUTORE\_TEMPORANEA.

Dato che la tabella AUTORE ora dovrebbe essere partizionata, è possibile verificarne la ridefinizione eseguendo una query su `USER_TAB_PARTITIONS`:

```
select Nome_Tabella, Nome_Tablespace, Valore_Superiore,
  from USER_TAB_PARTITIONS
  where Nome_Tabella = 'AUTORE';
```

NOME_TABELLA	NOME_TABLESPACE	VALORE_SUPERIORE
AUTORE	UTENTI	MAXVALUE
AUTORE	UTENTI	"M"

Come mostrato nel listato precedente, le partizioni della tabella AUTORE si trovano entrambe nella tablespace UTENTI. A questo punto, si dovrebbe controllare che le chiavi esterne nella tabella AUTORE siano state abilitate, che tutte le concessioni necessarie e gli indici siano presenti e che tutti i trigger siano stati abilitati.

## 6. Eliminare la tabella temporanea

Anche se AUTORE ha acquisito le definizioni di AUTORE\_TEMPORANEA, questa tabella esiste ancora! AUTORE e AUTORE\_TEMPORANEA si sono invertite di posto: infatti ora AUTORE\_TEMPORANEA è una tabella non partizionata. Dopo aver controllato che la tabella AUTORE sia corretta, si dovrebbe eliminare la tabella AUTORE\_TEMPORANEA, troncandola oppure eliminandola per liberare lo spazio in essa allocato.



## Capitolo 19

# Oracle e l'autorità

- 19.1 **Utenti, ruoli e privilegi**
- 19.2 **Che cosa possono concedere gli utenti**
- 19.3 **Concessione di risorse limitate**
- 19.4 **Opzioni avanzate**

L'informazioni sono vitali per il successo, ma quando sono danneggiate o si trovano in mani sbagliate, possono nascere gravi problemi. Oracle mette a disposizione caratteristiche di sicurezza estese per proteggere le informazioni da visualizzazioni non autorizzate e da danneggiamenti intenzionali o involontari. Questa sicurezza è fornita concedendo o revocando privilegi persona per persona e privilegio per privilegio, e si aggiunge (pur restando indipendente) a qualunque tecnologia di sicurezza già presente sul proprio sistema. Per controllare l'accesso ai dati, Oracle utilizza i comandi `create user`, `create role` e `grant`.

### 19.1 Utenti, ruoli e privilegi

Ogni utente Oracle ha un nome utente e una password e possiede le tabelle, le viste e le altre risorse da lui create. Un *ruolo* di Oracle è costituito da una serie di *privilegi* (o dal tipo di accesso di cui ciascun utente ha bisogno in base alla sua posizione e alle sue responsabilità). È possibile concedere privilegi particolari a dei ruoli e poi assegnare questi ruoli agli utenti appropriati. Un utente può anche concedere direttamente privilegi ad altri utenti.

*I privilegi di sistema del database* consentono di eseguire gruppi specifici di comandi. Il privilegio `CREATE TABLE`, per esempio, consente di creare delle tabelle, mentre il privilegio `GRANT ANY PRIVILEGE` permette di concedere qualsiasi privilegio di sistema.

*I privilegi di oggetti del database* offrono la possibilità di eseguire alcune operazioni su vari oggetti. Il privilegio `DELETE`, per esempio, consente di cancellare righe dalle tabelle e dalle viste; il privilegio `SELECT` consente invece di effettuare una query con un'istruzione `select` da tabelle, viste, sequenze e snapshot (viste materializzate).

Per un elenco completo dei privilegi di oggetti e di sistema, si rimanda alla voce “Privilegio” del Capitolo 42.

#### Creazione di un utente

Il sistema Oracle viene fornito con molti utenti già creati, tra cui `SYSTEM` e `SYS`. L'utente `SYS` possiede le tabelle interne più importanti che Oracle usa per gestire il database; `SYSTEM` invece possiede altre tabelle e viste. Per creare altri utenti, è possibile collegarsi all'utente `SYSTEM`, che ha questo privilegio.

Quando si installa Oracle, per prima cosa è necessario creare personalmente (o con l'aiuto di un amministratore di sistema) un utente per se stessi. Questa è la sintassi del comando `create user`:

```
create user utente identified {by password | externally};
```

Con questo comando è possibile impostare altri privilegi; per i dettagli, si rimanda alla voce relativa al comando `create user` del Capitolo 42.

Per connettere l'ID utente e la password del sistema del proprio computer alla sicurezza di Oracle, in modo che per gli utenti basati su host sia sufficiente un solo collegamento, occorre utilizzare “externally” invece di fornire una password. Un amministratore di sistema (che ha moltissimi privilegi) può desiderare l'ulteriore sicurezza di una password distinta. Negli esempi seguenti, l'amministratore del sistema viene chiamato Dora:

```
create user Dora identified by avocado;
```

Ora, l'account di Dora esiste ed è protetto da una password.

Inoltre, è possibile configurare l'utente con *tablespace* specifiche (spazi sul disco per le tabelle dell'utente, trattate nel capitolo seguente) e *quote* (limiti) per l'utilizzo di spazio e risorse. Per un'analisi delle *tablespace* e delle risorse, fare riferimento alla voce relativa al comando `create user` nei Capitoli 42 e 20.

Per modificare una password, occorre utilizzare il comando `alter user`:

```
alter user Dora identified by psyche;
```

Ora, Dora ha come password “psyche” invece di “avocado.” Tuttavia, Dora non può collegarsi al proprio account fino a quando non possiede il privilegio di sistema `CREATE SESSION`:

```
grant CREATE SESSION to Dora;
```

Più avanti in questo capitolo verranno proposti altri esempi sulle concessioni di privilegi di sistema.

## Gestione delle password

Le password possono scadere e gli account possono essere bloccati dopo ripetuti tentativi di connessione falliti. Quando si cambia la propria password, viene conservata una cronologia per evitare il riutilizzo di password precedenti.

Le caratteristiche di scadenza della password di un account sono determinate dal profilo assegnato all'account stesso. I profili, che vengono creati dal comando `create profile`, sono gestiti dal DBA (amministratore del database, analizzato più avanti in questo capitolo). Per ulteriori dettagli su questo comando, fare riferimento alla voce `CREATE PROFILE` nel Capitolo 42. Per quanto concerne le password e l'accesso all'account, i profili possono imporre quanto segue.

- La “durata” della password, che stabilisce con quale frequenza dev'essere modificata.
- Il periodo di proroga che segue la “data di scadenza” della password, durante il quale è possibile cambiare la password.
- Il numero di tentativi di connessione consecutivi falliti che sono consentiti prima che l'account venga automaticamente “bloccato”.

- Il numero di giorni in cui l'account rimane bloccato.
- Il numero di giorni che devono trascorrere prima di poter riutilizzare una password.
- Il numero di cambiamenti di password che devono verificarsi prima di poter riutilizzare una password.

Altre caratteristiche per la gestione delle password consentono di imporre una lunghezza e una complessità minime per le password.

Per modificare la password, oltre al comando `alter user` è possibile utilizzare il comando `password` in SQLPLUS. Se si usa questo comando, la nuova password non compare sullo schermo mentre la si digita. Gli amministratori di database possono modificare la password di qualsiasi altro utente con il comando `password`; gli altri utenti invece possono cambiare solo la propria password.

Quando si digita il comando `password`, vengono richieste la nuova e la vecchia password, come mostrato nel listato seguente:

```
connect dora/psyche
password
Changing password for dora
Old password:New password:
Retype new password:
```

Una volta che la password è stata cambiata con successo il sistema avvisa con il messaggio:

```
Password changed
```

### **Imposizione di una data di scadenza per la password**

Con i profili è possibile gestire la scadenza, il riutilizzo e la complessità delle password. Si può limitare la durata di una password, e bloccare un account la cui password è ormai troppo obsoleta. È inoltre possibile imporre una complessità moderata per una password, e bloccare qualunque account presenti ripetuti tentativi di connessione falliti.

Per esempio, se si imposta la risorsa `FAILED_LOGIN_ATTEMPTS` del profilo dell'utente a 5, per l'account saranno ammessi quattro tentativi consecutivi di connessione falliti; al quinto tentativo l'account verrà bloccato.

**NOTA** *Se al quinto tentativo viene inserita la password corretta, il conteggio dei tentativi di connessione falliti viene ripristinato a 0, dando la possibilità di fallire altri cinque tentativi consecutivi di connessione prima di bloccare l'account.*

Nel listato seguente viene creato il profilo `LIMITED_PROFILE` che dovrà essere utilizzato dall'utente JANE:

```
create profile LIMITED_PROFILE limit
FAILED_LOGIN_ATTEMPTS 5;

create user JANE identified by EYRE
profile LIMITED_PROFILE;

grant CREATE SESSION to JANE;
```

Se si verificano cinque connessioni consecutive fallite all'account JANE, Oracle bloccherà automaticamente l'account. Quando poi si utilizzerà la password corretta per l'account JANE, comparirà il messaggio di errore seguente:

```
connect jane/eyre
ERROR: ORA-28000: the account is locked
```

Per sbloccare l'account, è necessario ricorrere alla clausola `account unlock` del comando `alter user` (dall'account di un DBA), come mostrato nel listato seguente:

```
alter user JANE account unlock;
```

Una volta sbloccato l'account, sarà di nuovo possibile connettersi all'account JANE. Con la clausola `account lock` del comando `alter user` è possibile bloccare un account manualmente.

```
alter user JANE account lock;
```

Se un account si blocca a causa dei ripetuti tentativi di connessione falliti, si sbloccherà automaticamente non appena il valore `PASSWORD_LOCK_TIME` del suo profilo viene superato. Per esempio, se `PASSWORD_LOCK_TIME` fosse impostato a 1, l'account JANE dell'esempio precedente rimarrebbe bloccato per un giorno, trascorso il quale si sbloccherebbe.

Con la risorsa `PASSWORD_LIFE_TIME` dei profili è possibile stabilire una durata massima per una password. Per esempio, si potrebbe costringere gli utenti del profilo `LIMITED_PROFILE` a modificare le proprie password ogni 30 giorni.

```
alter profile LIMITED_PROFILE limit
PASSWORD_LIFE_TIME 30;
```

In questo esempio, il comando `alter profile` viene utilizzato per modificare il profilo `LIMITED_PROFILE`. Il valore `PASSWORD_LIFE_TIME` viene impostato a 30, quindi la password di qualsiasi account utilizzi questo profilo scadrà dopo 30 giorni. Se la password è scaduta, la si dovrà cambiare la prima volta che si cerca di connettersi, a meno che il profilo non preveda un periodo di proroga per le password scadute. Il parametro del periodo di proroga prende il nome di `PASSWORD_GRACE_TIME`. Se la password non viene modificata entro questo periodo, l'account scade.

**NOTA** *Se si intende utilizzare il parametro `PASSWORD_LIFE_TIME`, allora si dovrà concedere agli utenti la possibilità di modificare senza problemi le proprie password.*

Un account “scaduto” è diverso da uno “bloccato”. Un account bloccato, come si è visto in precedenza, si potrebbe sbloccare automaticamente con il passare del tempo, mentre invece un account scaduto richiede l'intervento manuale di un DBA per essere nuovamente attivato.

**NOTA** *Se si utilizzano le caratteristiche di scadenza delle password, è opportuno accertarsi che gli account che possiedono le applicazioni abbiano impostazioni di profilo differenti, altrimenti potrebbero bloccarsi, rendendo inutilizzabile l'applicazione.*

Per attivare nuovamente un account scaduto, si dovrà eseguire il comando `alter user`, come mostrato nell'esempio seguente. In questo esempio, il DBA fa scadere manualmente la password di JANE:

```
alter user jane password expire;
```

User altered.

Successivamente, JANE cerca di connettersi al proprio account. Quando inserisce la password, le viene richiesta immediatamente una nuova password per l'account.

```
connect jane/eyre
ERROR: ORA-28001: the account has expired
```

```
Changing password for jane
Old password:
New password:
Retype new password:
Password changed
Connected.
```

Si può anche obbligare gli utenti a modificare le password la prima volta che accedono ai propri account, grazie alla clausola `password expire` del comando `create user`. Tuttavia, il comando `create user` non consente di impostare una data di scadenza per la nuova password impostata dall'utente; per poterlo fare, si devono utilizzare i parametri del profilo `PASSWORD_LIFETIME`, mostrati negli esempi precedenti.

Per visualizzare la data di scadenza della password di qualsiasi account, è necessario eseguire una query sulla colonna `Data_Scadenza` della vista del dizionario dati `DBA_USERS`. Gli utenti che desiderano visualizzare la data di scadenza della password dei propri account possono eseguire una query sulla colonna `Data_Scadenza` della vista del dizionario dati `USER_USERS` (con SQL\*Plus oppure con uno strumento di query basato su client).

### **Imposizione di limiti al riutilizzo di una password**

Per evitare che una password venga riutilizzata, si può utilizzare uno di questi due parametri di profilo: `PASSWORD_REUSE_MAX` o `PASSWORD_REUSE_TIME`. Questi due parametri si escludono a vicenda; se si imposta un valore per uno di essi, l'altro dovrà essere impostato a `UNLIMITED`.

Il parametro `PASSWORD_REUSE_TIME` specifica il numero di giorni che devono trascorrere prima di poter riutilizzare una password. Per esempio, se si imposta `PASSWORD_REUSE_TIME` a 60, la stessa password non potrà essere riutilizzata per 60 giorni.

Il parametro `PASSWORD_REUSE_MAX` specifica invece il numero di modifiche di password che devono verificarsi prima che una password possa essere riutilizzata. Se si cerca di riutilizzare la password prima che il limite sia stato raggiunto, Oracle rifiuterà la modifica apportata alla password.

Per esempio, è possibile impostare `PASSWORD_REUSE_MAX` per il profilo `LIMITED_PROFILE` creato in precedenza.

```
alter profile LIMITED_PROFILE limit
PASSWORD_REUSE_MAX 3
PASSWORD_REUSE_TIME UNLIMITED;
```

Se ora l'utente JANE cerca di riutilizzare una password recente, il tentativo di modificare la password non riuscirà. Per esempio, si supponga che JANE modifichi la propria password, come nella linea seguente:

```
alter user JANE identified by austen;
```

e poi la cambi ancora:

```
alter user JANE identified by whitley;
```

Durante il tentativo di modifica successivo, cerca di riutilizzare una password recente, ma il tentativo fallisce.

```

alter user jane identified by austen;
alter user jane identified by austen
*
ERROR at line 1:
ORA-28007: the password cannot be reused

```

JANE non può riutilizzare nessuna delle password recenti e dovrà quindi inventarsene una nuova.

Le cronologie delle password sono memorizzate in una tabella di nome USER\_HISTORY\$ nello schema SYS. In questa tabella Oracle memorizza l'ID utente, il valore della password cifrato e la data/indicatore orario per la creazione della password. Quando si supera il valore PASSWORD\_REUSE\_TIME, oppure il numero di modifiche supera PASSWORD\_REUSE\_MAX, i record delle vecchie password vengono cancellati dalla tabella SYS.USER\_HISTORY\$. Se una nuova cifratura corrisponde a una già esistente, la nuova password non viene accettata.

Dato che le vecchie password sono memorizzate in una tabella posseduta da SYS, i dati verranno memorizzati nella tablespace di SYSTEM. Di conseguenza, se si decide di mantenere una cronologia di password molto grande per molti utenti che sono costretti a modificare le password frequentemente, i requisiti di spazio per la tabella con la cronologia delle password (SYS.USER\_HISTORY\$) potrebbero influire su quelli della tablespace di SYSTEM.

## Tre ruoli standard

Ora che Dora ha un account, che cosa può fare in Oracle? A questo punto ancora nulla, in quanto non ha alcun privilegio di sistema.

Oracle mette a disposizione tre ruoli standard per la compatibilità con le versioni precedenti: CONNECT, RESOURCE e DBA.

### Il ruolo CONNECT

Agli utenti occasionali in genere viene assegnato solo il ruolo CONNECT o il privilegio CREATE SESSION. CONNECT dà agli utenti la possibilità di connettersi e di eseguire le funzioni base. Questo diritto diventa significativo con l'aggiunta dell'accesso a tabelle specifiche che appartengono ad altri utenti e con la concessione dei privilegi per selezionare, inserire, aggiornare e cancellare righe in queste tabelle. Oltre alle sessioni, gli utenti che hanno il ruolo CONNECT possono anche creare tabelle, viste, sequenze, cluster, sinonimi (trattati più avanti), e collegamenti ad altri database (trattati nel Capitolo 22).

### Il ruolo RESOURCE

A utenti del database più esigenti e regolari può essere concesso il ruolo RESOURCE. RESOURCE concede agli utenti anche i diritti di creare proprie tabelle, sequenze, procedure, trigger, tipi di dati (fare riferimento alla Parte quarta di questo libro), operatori, tipi di indici, indici e cluster. Per una discussione sulle stored procedure e i trigger si rimanda alla Parte terza di questo volume.

### Il ruolo DBA

Il ruolo DBA (*amministratore del database*) ha tutti i privilegi di sistema, comprese quote di spazio illimitate, e la possibilità di concedere tutti i privilegi ad altri utenti. In questo capitolo, dba si riferisce all'amministratore del database e ha un ruolo DBA, mentre DBA si riferisce solo ai privilegi inclusi nel ruolo DBA. SYSTEM serve per essere utilizzato da un utente DBA. Alcuni dei diritti riservati a un dba non sono mai assegnati, o richiesti, agli utenti normali. Questi diritti

verranno quindi analizzati molto brevemente. Altri diritti normalmente utilizzati dai dba sono impiegati regolarmente anche dagli utenti e sono importanti. Questo sottoinsieme di privilegi DBA sarà presentato tra breve.

## Sintassi del comando grant

La sintassi del comando grant per i privilegi di sistema è la seguente:

```
grant {privilegio sistema | ruolo | all [privileges] }
  [. {privilegio sistema | ruolo | all [privileges]}
  to {utente | ruolo} [. {utente | ruolo}]
  [identified by password ]
  [with admin option]
```

Utilizzando il comando grant è possibile concedere qualunque privilegio o ruolo di sistema a un altro utente, a un altro ruolo, oppure a public. La clausola with admin option consente al beneficiario di assegnare il privilegio o ruolo ad altri utenti o ruoli. La clausola all concede all'utente o al ruolo tutti i privilegi eccetto il privilegio di sistema SELECT ANY DICTIONARY.

Chi ha concesso il ruolo a un utente può anche revocarlo (revoke).

## Revoca di privilegi

I privilegi concessi possono anche essere annullati. Il comando revoke è simile al comando grant:

```
revoke {privilegio sistema | ruolo | all [privileges]}
  [. {privilegio sistema | ruolo | all [privileges]}].
  from {utente | ruolo} [. {utente | ruolo}].
```

Una persona con ruolo DBA può revocare CONNECT, RESOURCE, DBA o qualsiasi altro privilegio o ruolo a chiunque altro, compreso un altro dba. Ciò naturalmente è pericoloso, quindi i privilegi DBA dovrebbero essere concessi con cautela e solo a una stretta minoranza di persone per cui sono realmente necessari.

**NOTA** *Se a un utente vengono revocate tutte le concessioni, esso non viene comunque eliminato da Oracle, né vengono distrutte le tabelle da lui create; gli viene semplicemente proibito di accedervi. Altri utenti con accesso alle tabelle possono accedervi esattamente come prima.*

Per rimuovere un utente e tutte le risorse da lui possedute, occorre utilizzare il comando drop user:

```
drop user nomeutente [cascade];
```

L'opzione cascade rimuove l'utente con tutti gli oggetti da lui posseduti, compresi i vincoli di integrità referenziale. Questa opzione rende non più validi viste, sinonimi, stored procedure, funzioni o package che si riferiscono a oggetti presenti nello schema dell'utente rimosso. Se non si usa l'opzione cascade ed esistono ancora oggetti posseduti dall'utente, Oracle non rimuove l'utente e restituisce invece un messaggio di errore.

## 19.2 Che cosa possono concedere gli utenti

Un utente può concedere privilegi su qualsiasi oggetto da lui posseduto. Il dba può concedere qualunque privilegio di sistema (perché il ruolo DBA possiede i privilegi GRANT ANY e GRANT ANY ROLE).

Si supponga che l'utente Dora possieda la tabella COMFORT e sia un dba. Dora crea due nuovi utenti, Bob e Judy, con i seguenti privilegi:

```
create user Judy identified by sarah;
```

User created.

```
grant CONNECT to Judy;
```

Grant succeeded.

```
create user Bob identified by carolyn;
```

User created.

```
grant CONNECT, RESOURCE to bob;
```

Grant succeeded.

Questa sequenza di comandi consente sia a Judy sia a Bob di connettersi a Oracle, e concede a Bob qualche capacità in più. Ma questi utenti possono fare qualcosa con le tabelle di Dora? Non senza un accesso esplicito.

Per concedere ad altri utenti l'accesso alle proprie tabelle, occorre fare ricorso a una seconda forma del comando grant:

```
grant { privilegio oggetto | all [privileges] }
[(colonna [, colonna] . . .)]
[. { privilegio oggetto | all [privileges] }
[(colonna [, colonna] . . .)] ] . .
on oggetto to {utente | ruolo}
[with grant option]
[with hierarchy option];
```

Ecco alcuni dei privilegi che un utente può concedere.

- Sulle tabelle, viste e viste materializzate dell'utente: INSERT  
UPDATE (tutte o solo alcune colonne specifiche)  
DELETE  
SELECT.

I privilegi INSERT, UPDATE e DELETE possono essere concessi sulle viste materializzate solo se queste sono aggiornabili. Per dettagli sulla creazione delle viste materializzate, si rimanda al Capitolo 23.

- Sulle tabelle si possono garantire anche:  
ALTER (tabella, tutte o solo alcune colonne specifiche)  
REFERENCES  
INDEX (colonne in una tabella)

## ON COMMIT REFRESH

### QUERY REWRITE

ALL (tutti gli elementi elencati in precedenza)

- Sulle procedure, funzioni, package, tipi di dati astratti, librerie, tipi di indici e operatori: EXECUTE

- Sulle sequenze:

SELECT

ALTER

- Sulle directory (per i tipi di dati BFILE LOB e le tabelle esterne):

READ

WRITE

- Sui tipi di dati astratti e viste:

UNDER (per la capacità di creare viste secondate in una vista, oppure tipi secondari in un tipo)

Il privilegio concesso dev'essere uno dei privilegi oggetto (ALL, ALTER, DELETE, EXECUTE, INDEX, INSERT, ON COMMIT REFRESH, QUERY REWRITE, READ, REFERENCES, SELECT, UNDER, UPDATE o WRITE). Questi privilegi offrono al beneficiario della concessione la possibilità di eseguire alcune azioni sugli oggetti. L'oggetto può essere uno degli oggetti elencati qui, oppure un sinonimo di uno qualsiasi di questi oggetti.

Quando si esegue una procedura o una funzione di un altro utente, in genere la si esegue utilizzando i privilegi del suo proprietario. Questo significa che non è necessario un accesso esplicito ai dati utilizzati dalla procedura o funzione; si vedono solo i risultati dell'esecuzione, non i dati di base. Inoltre, si possono creare stored procedure che vengono eseguite con i privilegi dell'utente invocante e non del proprietario della procedura; per i dettagli si rimanda al Capitolo 29.

Dora concede a Bob l'accesso SELECT alla tabella COMFORT:

```
grant select on COMFORT to Bob;
```

```
Grant succeeded.
```

La clausola with grant option del comando grant permette a chi riceve la concessione di trasmettere i privilegi ricevuti ad altri utenti. Se Dora concede privilegi with grant option sulle sue tabelle all'utente Bob, quest'ultimo può fare concessioni sulle tabelle di Dora ad altri utenti (Bob può solo trasmettere i privilegi che gli sono stati concessi, come SELECT). Se si ha intenzione di creare delle viste basate sulle tabelle di un altro utente, e di concedere l'accesso a queste viste ad altri utenti, innanzitutto è necessario avere un accesso with grant option alle tabelle di base.

## Passaggio a un altro utente con connect

Per controllare la riuscita del comando grant, Dora può connettersi al nome utente di Bob con il comando connect. Questo comando può essere utilizzato con uno dei metodi seguenti:

- Inserendo il nome utente e la password sulla stessa linea del comando.
- Digitando il comando da solo e poi rispondendo alle richieste.
- Digitando il comando e il nome utente e poi rispondendo alla richiesta della password.

Gli ultimi due metodi evitano la visualizzazione della password e pertanto sono più sicuri. Il listato seguente mostra una connessione di esempio al database.

```
connect Bob/carolyn
```

```
Connected.
```

Una volta connessa, Dora può effettuare selezioni dalla tabella su cui ha concesso a Bob l'accesso SELECT.

**NOTA** *A meno che non si utilizzi un sinonimo, il nome della tabella dev'essere preceduto dal nome utente del proprietario, altrimenti Oracle comunica che la tabella non esiste.*

```
select * from Dora.COMFORT;
```

CITTA	DATA	CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
SAN FRANCISCO	21-MAR-01	16.9	5.7	.5	
SAN FRANCISCO	22-JUN-01	10.6	22.1	.1	
SAN FRANCISCO	23-SEP-01		16.4	.1	
SAN FRANCISCO	22-DEC-01	11.4	4.3	2.3	
KEENE	21-MAR-01	4.4	-18.4	4.4	
KEENE	22-JUN-01	29.5	19.2	1.3	
KEENE	23-SEP-01	37.6	28.1		
KEENE	22-DEC-01	-21.7	-18.4	3.9	

Per comodità viene creata una vista chiamata COMFORT, che è una semplice selezione completa della tabella Dora.COMFORT:

```
create view COMFORT as select * from Dora.COMFORT;
```

```
View created.
```

Una selezione da questa vista produce esattamente gli stessi risultati di una selezione dalla tabella Dora.COMFORT:

```
select * from COMFORT;
```

CITTA	DATA	CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
SAN FRANCISCO	21-MAR-01	16.9	5.7	.5	
SAN FRANCISCO	22-JUN-01	10.6	22.1	.1	
SAN FRANCISCO	23-SEP-01		16.4	.1	
SAN FRANCISCO	22-DEC-01	11.4	4.3	2.3	
KEENE	21-MAR-01	4.4	-18.4	4.4	
KEENE	22-JUN-01	29.5	19.2	1.3	
KEENE	23-SEP-01	37.6	28.1		
KEENE	22-DEC-01	-21.7	-18.4	3.9	

Ora, Dora ritorna al proprio nome utente e crea una vista che seleziona soltanto una parte della tabella COMFORT:

```
connect Dora/psyche
Connected.
```

```
create view QUALCHECOMFORT as
select * from COMFORT
where Citta = 'KEENE';
```

View created.

Successivamente concede a Bob i privilegi SELECT e UPDATE su questa vista e gli revoca (attraverso il comando revoke) tutti i privilegi sulla tabella COMFORT:

```
grant select, update on QUALCHECOMFORT to Bob;
```

Grant succeeded.

```
revoke all on COMFORT from Bob;
```

Revoke succeeded.

Dora poi si connette di nuovo al nome utente di Bob per verificare gli effetti di questo cambiamento:

```
connect Bob/carolyn
Connected.
```

```
select * from COMFORT;
*
```

```
ERROR at line 1: ORA-04063: view "BOB.COMFORT" has errors
```

Il tentativo di effettuare selezioni dalla vista COMFORT non riesce, perché la tabella di base per questa vista di Bob è Dora.COMFORT. Ovviamente, un tentativo di selezione dalla tabella Dora.COMFORT produce esattamente lo stesso messaggio. Poi, viene effettuato un tentativo di selezione da Dora.QUALCHECOMFORT:

```
select * from Dora.QUALCHECOMFORT;
```

CITTA	DATACAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
KEENE	21-MAR-01	4.4	-18.4	4.4
KEENE	22-JUN-01	29.5	19.2	1.3
KEENE	23-SEP-01	37.6	28.1	
KEENE	22-DEC-01	-21.7	-18.4	3.9

Questa query funziona perfettamente, anche se l'accesso diretto alla tabella COMFORT era stato revocato, perché Dora ha concesso a Bob l'accesso a una parte di COMFORT attraverso la vista QUALCHECOMFORT. È solo la parte della tabella relativa a KEENE.

Questo esempio mostra una potente caratteristica di sicurezza di Oracle: è possibile creare viste utilizzando praticamente qualsiasi restrizione si desideri o qualsiasi calcolo nelle colonne, e poi concedere ad altri utenti l'accesso alla vista, invece che alle tabelle di base. Gli utenti sono in grado di vedere solo le informazioni presentate dalla vista. Il meccanismo può essere esteso in modo che sia specifico per ogni utente. Più avanti in questo capitolo, nel paragrafo “Sicurezza con User” verrà fornita una spiegazione dettagliata di questa caratteristica.

Ora viene creata la vista POCOCOMFORT sulla vista QUALCHECOMFORT, sotto il nome utente di Bob:

```
create view POCOCOMFORT as select * from Dora.QUALCHECOMFORT;
```

View created.

e la riga per il 23.09.01 viene aggiornata:

```
update POCOCOMFORT set Mezzogiorno = 31.1
where DataCampione = '23-SEP-01';
```

1 row updated.

Quando viene effettuata una query sulla vista POCOCOMFORT, saranno visibili gli effetti dell'aggiornamento:

```
select * from POCOCOMFORT;
```

CITTA	DATA CAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
KEENE	21-MAR-01	4.4	-18.4	4.4
KEENE	22-JUN-01	29.5	19.2	1.3
KEENE	23-SEP-01	31.1	28.1	
KEENE	22-DEC-01	-21.7	-18.4	3.9

Una query di Dora.QUALCHECOMFORT mostrerebbe gli stessi risultati, come anche una query di COMFORT effettuata da Dora nel proprio nome utente. L'aggiornamento si è avuto anche sulla tabella di base, nonostante sia stato effettuato attraverso due viste (POCOCOMFORT e QUALCHECOMFORT).

**NOTA** È necessario concedere agli utenti l'accesso **SELECT** a qualsiasi tabella in cui possono aggiornare o cancellare record. Questa pratica è conforme allo standard ANSI che si sta sviluppando e riflette il fatto che un utente che ha avuto solo privilegi **UPDATE** o **DELETE** su una tabella potrebbe utilizzare i commenti di ritorno del database per scoprire informazioni sui dati di base.

## create synonym

Un metodo alternativo alla creazione di una vista che include un'intera tabella o una vista di un altro utente consiste nel creare un sinonimo:

```
create synonym POCOCOMFORT for Dora.QUALCHECOMFORT;
```

Questo sinonimo può essere trattato esattamente come una vista. Per informazioni, si rimanda al comando **create synonym** nel Capitolo 42.

## Uso di privilegi non concessi

Si supponga che venga effettuato un tentativo di cancellare la riga che è appena stata aggiornata:

```
delete from POCOCOMFORT where DataCampione = '23-SEP-01';
*
```

ERROR at line 1: ORA-01031: insufficient privileges

A Bob non sono stati concessi da Dora i privilegi **DELETE**, pertanto il tentativo non riesce.

## Trasmissione di privilegi

Bob può concedere ad altri utenti l'autorità per accedere alle sue tabelle, ma non può assegnare ad altri l'accesso a tabelle che non gli appartengono. In questo esempio, tenta di concedere a Judy l'autorità INSERT:

```
grant insert on Dora.QUALCHECOMFORT to Judy;
*
ERROR at line 1: ORA-01031: insufficient privileges
```

Dato che Bob non ha questa autorità, non riesce a trasmetterla a Judy. Successivamente, Bob cerca di trasmettere un privilegio di cui dispone, SELECT:

```
grant select on Dora.QUALCHECOMFORT to Judy;
*
ERROR at line 1: ORA-01031: insufficient privileges
```

Non può concedere neanche questo privilegio, perché la vista QUALCHECOMFORT non gli appartiene. Se gli fosse stato concesso l'accesso a QUALCHECOMFORT with grant option, il comando grant precedente avrebbe avuto successo. La vista POCOCOMFORT gli appartiene, quindi Bob può cercare di trasmettere l'autorità su questa a Judy:

```
grant select on POCOCOMFORT to Judy;

ERROR at line 1:
ORA-01720: grant option does not exist for 'DORA.QUALCHECOMFORT'
```

Dato che POCOCOMFORT si basa su una delle viste di Dora, e a Bob non è stato concesso SELECT with grant optino su questa vista, la concessione di Bob non riesce.

In aggiunta viene creata una nuova tabella posseduta da Bob e caricata con le informazioni correnti della vista POCOCOMFORT:

```
create table NOCOMFORT as
select * from POCOCOMFORT;
```

Table created.

Poi vengono concessi anche a Judy i privilegi SELECT su questa tabella:

```
grant select on NOCOMFORT to Judy;

Grant succeeded.
```

Dall'account di Judy vengono effettuate delle query sulla tabella per la quale Bob ha concesso a Judy il privilegio SELECT:

```
connect Judy/sarah
select * from Bob.NOCOMFORT;
```

CITTA	DATACAMPIONE	MEZZOGIORNO	MEZZANOTTE	PRECIPITAZIONE
KEENE	21-MAR-01	4.4	-18.4	4.4
KEENE	22-JUN-01	29.5	19.2	1.3
KEENE	23-SEP-01	31.1	28.1	
KEENE	22-DEC-01	-21.7	-18.4	3.9

Questa concessione è riuscita. È stata creata una tabella NOCOMFORT dal nome utente di Bob, che quindi la possiede. Egli è stato quindi in grado di concedere l'accesso a questa tabella.

Se Dora vuole che Bob possa trasmettere i suoi privilegi ad altri, può aggiungere un'altra clausola all'istruzione grant:

```
connect Dora/psyche
grant select, update on QUALCHECOMFORT to Bob with grant option;
Grant succeeded.
```

La clausola with grant option consente a Bob di trasmettere a Judy l'accesso a QUALCHECOMFORT, attraverso la vista POCOCOMFORT. Si osservi che i tentativi di accesso a una tabella per cui l'utente non ha il privilegio SELECT, producono sempre questo messaggio:

```
ERROR at line 1: ORA-00942: table or view does not exist
```

Questo messaggio che appare non fa alcun cenno al fatto che non si ha il privilegio di accesso; in tal modo un utente che non ha il permesso di effettuare una query su una tabella non saprà neppure che questa esiste.

## **Creazione di un ruolo**

Oltre ai tre ruoli di sistema mostrati in precedenza, CONNECT, RESOURCE e DBA, in Oracle esiste la possibilità di creare ruoli propri. I ruoli creati possono comprendere privilegi di tabelle o di sistema o una combinazione di entrambi. I paragrafi seguenti mostrano come creare e amministrare i ruoli.

Per creare un ruolo, è necessario avere il privilegio di sistema CREATE ROLE. La sintassi per la creazione di un ruolo è mostrata nel listato seguente:

```
create role nome_ruolo
[not identified
|identified {by password | using [schema.]package
|externally | globally }];
```

Un ruolo appena creato non è associato ad alcun privilegio. Le opzioni di password per i ruoli sono trattate nel paragrafo “Aggiunta di una password a un ruolo” più avanti nel capitolo.

Ecco due esempi di comandi create role:

```
create role IMPIEGATO;
create role MANAGER;
```

Il primo comando crea un ruolo denominato IMPIEGATO, che verrà usato negli esempi dei prossimi paragrafi di questo libro. Il secondo comando crea un ruolo denominato MANAGER, anch'esso utilizzato in questi esempi.

## **Concessione di privilegi a un ruolo**

Una volta che il ruolo è stato creato, è possibile concedergli dei privilegi. La sintassi del comando grant per i ruoli è uguale a quella per gli utenti. Quando si concedono privilegi a dei ruoli, occorre utilizzare il nome del ruolo nella clausola to del comando grant, come mostrato nel listato seguente:

---

```
grant select on COMFORT to IMPIEGATO;
```

Come mostrato in questo esempio, il nome del ruolo sostituisce nel comando grant il nome utente. Il privilegio di effettuare selezioni dalla tabella COMFORT è ora disponibile per tutti gli utenti del ruolo IMPIEGATO.

Un dba, o chi ha il ruolo di sistema GRANT ANY PRIVILEGE, può concedere ai ruoli privilegi di sistema come CREATE SESSION, CREATE SYNONYM e CREATE VIEW. Questi privilegi saranno quindi utilizzabili da ogni utente del ruolo.

La possibilità di collegarsi al database è concessa attraverso il privilegio CREATE SESSION. Nell'esempio seguente, questo privilegio viene concesso al ruolo IMPIEGATO. Questo privilegio viene concesso anche al ruolo MANAGER, insieme al privilegio di sistema CREATE DATABASE LINK.

```
grant CREATE SESSION to IMPIEGATO;
grant CREATE SESSION, CREATE DATABASE LINK to MANAGER;
```

## **Concessione di un ruolo a un altro ruolo**

I ruoli possono essere concessi ad altri ruoli. Ciò è possibile grazie al comando grant, come mostrato nell'esempio seguente:

```
grant IMPIEGATO to MANAGER;
```

In questo esempio, il ruolo IMPIEGATO viene concesso al ruolo MANAGER. Anche se non si è concesso direttamente alcun privilegio di tabella al ruolo MANAGER, esso eredita tutti i privilegi concessi al ruolo IMPIEGATO. La disposizione dei ruoli in questo modo è abbastanza comune nelle organizzazioni gerarchiche.

Quando si concede un ruolo a un altro ruolo (o ad un utente, come viene spiegato nel paragrafo seguente) si può utilizzare la clausola with admin option, come illustra il listato seguente:

```
grant IMPIEGATO to MANAGER with admin option;
```

Se si usa questa clausola, il destinatario della concessione ha l'autorità di concedere il ruolo ad altri utenti o ruoli. Il destinatario può anche alterare o rimuovere il ruolo stesso.

## **Concessione di un ruolo a utenti**

È possibile concedere dei ruoli agli utenti. In questo caso, i ruoli possono essere considerati come insiemi di privilegi cui è stato assegnato un nome. Invece di concedere ciascun privilegio a ciascun utente, si concedono i privilegi al ruolo e poi il ruolo ad ogni utente. Questo semplifica enormemente i compiti amministrativi coinvolti nella gestione dei privilegi.

**NOTA** *I privilegi concessi agli utenti attraverso ruoli non possono essere utilizzati come base per viste, procedure, funzioni, package o chiavi esterne. Quando si creano questi tipi di oggetti del database, occorre concedere direttamente i privilegi necessari.*

È possibile concedere un ruolo a un utente attraverso il comando grant, come mostrato nell'esempio seguente:

```
grant IMPIEGATO to Bob;
```

In questo esempio, l'utente Bob ha tutti i privilegi concessi al ruolo IMPIEGATO (ossia i privilegi CREATE SESSION e SELECT su COMFORT).

Quando si concede un ruolo a un utente, si può utilizzare la clausola with admin option, come mostrato nel listato seguente:

```
grant MANAGER to Dora with admin option;
```

Dora ha ora l'autorità di concedere il ruolo MANAGER ad altri utenti o ruoli, oppure di modificare e rimuovere il ruolo.

## Aggiunta di una password a un ruolo

Il comando alter role può essere utilizzato per un solo scopo: modificare l'autorità necessaria a concederlo. Per default, ai ruoli non è associata alcuna password. Per consentire misure di sicurezza per un ruolo, occorre utilizzare la parola chiave identified nel comando alter role. Esistono due modi per implementare questa sicurezza.

Innanzitutto è possibile utilizzare la clausola identified by del comando alter role per specificare una password, come mostrato nel listato seguente:

```
alter role MANAGER identified by cygnusxi;
```

Ogni volta che un utente cerca di attivare questo ruolo, viene richiesta la password. Se tuttavia questo ruolo è impostato come ruolo predefinito per l'utente, quando l'utente si connette, non viene richiesta alcuna password. Per ulteriori dettagli su questi argomenti si rimanda al paragrafo “Attivazione e disattivazione di ruoli” di questo capitolo.

I ruoli possono anche essere legati ai privilegi del sistema operativo. Se il sistema operativo prevede questa possibilità, allora si può utilizzare la clausola identified externally del comando alter role. Quando il ruolo viene attivato, Oracle controlla il sistema operativo per verificare l'accesso. Nell'esempio seguente viene mostrato come modificare il ruolo per sfruttare questa caratteristica di sicurezza:

```
alter role MANAGER identified externally;
```

Nella maggior parte dei sistemi UNIX il processo di verifica impiega il file denominato /etc/group. Per utilizzare questo file in qualsiasi sistema operativo, il parametro OS\_ROLES di avvio del database nel file init.ora dovrà essere impostato a TRUE.

L'esempio seguente mostra questo processo di verifica per un'istanza del database chiamata “Local” su un sistema UNIX. Il file /etc/group del server può contenere la voce seguente:

```
ora_local_manager_d:NONE:1:dora
```

Questa voce concede il ruolo MANAGER all'account Dora. Il suffisso “\_d” indica che questo ruolo deve essere concesso per default quando Dora si connette. Un suffisso “\_a” avrebbe indicato che questo ruolo doveva essere attivato with admin option. Se questo ruolo fosse anche quello predefinito per l'utente, il suffisso sarebbe stato “\_ad”. Se questo ruolo fosse stato concesso a più di un utente, si sarebbero dovuti aggiungere i nomi utente alla voce di /etc/group, come mostrato nel listato seguente:

```
ora_local_manager_d:NONE:1:dora,judy
```

Se si applica questa opzione, tutti i ruoli nel database verranno attivati attraverso il sistema operativo.

## Rimozione di una password da un ruolo

Per rimuovere una password da un ruolo, occorre utilizzare la clausola `not identified` del comando `alter role`, come mostrato nel listato seguente. Per default, i ruoli non hanno password.

```
alter role MANAGER not identified;
```

## Attivazione e disattivazione di ruoli

Quando si modifica l'account di un utente, è possibile creare un elenco di ruoli predefiniti per questo utente, attraverso la clausola `default role` del comando `alter user`. L'azione predefinita di questo comando imposta tutti i ruoli di un utente come ruoli predefiniti, attivandoli tutti ogni volta che l'utente si connette.

**NOTA** *Il numero massimo di ruoli che si possono attivare per un utente viene impostato con il parametro di inizializzazione del database `MAX_ENABLED_ROLES`.*

La sintassi per questa parte del comando `alter user` è la seguente:

```
alter user nameutente
default role {[ruolo1, ruolo2]
[all|all except ruolo1, ruolo2][NONE]};
```

Come mette in luce questa sintassi, un utente può essere modificato in modo da avere attivati, per default, ruoli specifici, tutti i ruoli, tutti i ruoli tranne quelli specifici oppure nessun ruolo. Per esempio, il seguente comando `alter user` attiva il ruolo `IMPIEGATO` ogni volta che Bob si connette:

```
alter user Bob
default role IMPIEGATO;
```

Per attivare un ruolo non predefinito, occorre utilizzare il comando `set role`, come mostrato in questo esempio:

```
set role IMPIEGATO;
```

Si possono utilizzare anche le clausole `all` e `all except` che erano disponibili nel comando `alter user`:

```
set role all;
set role all except IMPIEGATO;
```

Se un ruolo è associato a una password, questa dev'essere specificata attraverso una clausola `identified by`:

```
set role MANAGER identified by cygnusxi;
```

Per disattivare un ruolo nella propria sessione, occorre utilizzare il comando `set role none`, come mostrato nel listato seguente. Questo disattiva tutti i ruoli nella sessione corrente. Dopo che tutti i ruoli sono stati disattivati, occorre attivare di nuovo quelli che si desiderano.

```
set role none;
```

Dato che potrebbe essere necessario eseguire ogni tanto un comando `set role none`, è preferibile concedere agli utenti il privilegio `CREATE SESSION` direttamente invece che attraverso i ruoli.

## Revoca di privilegi da un ruolo

Per revocare un privilegio da un ruolo, occorre utilizzare il comando `revoke` descritto in precedenza in questo capitolo, specificando il privilegio, il nome dell'oggetto (se si tratta di un privilegio di oggetto) e il nome del ruolo, come mostrato nell'esempio seguente:

```
revoke SELECT on COMFORT from IMPIEGATO;
```

Ora, gli utenti del ruolo `IMPIEGATO` non possono più effettuare query sulla tabella `COMFORT`.

## Eliminazione di un ruolo

Per eliminare un ruolo, occorre utilizzare il comando `drop role`, come mostrato nell'esempio seguente:

```
drop role MANAGER;
drop role IMPIEGATO;
```

I ruoli specificati, e i privilegi loro associati, verranno rimossi completamente dal database.

## Concessione di update a colonne specifiche

In alcuni casi, si potrebbe concedere agli utenti il privilegio `SELECT` su più colonne di quelle su cui si desidera concedere il privilegio `UPDATE`. Dato che le colonne di `SELECT` possono essere ristrette attraverso una vista, per limitare ulteriormente le colonne che possono essere aggiornate è necessaria una forma particolare del comando `grant` dell'utente. Ecco un esempio per due colonne nella tabella `COMFORT`:

```
grant update (Mezzogiorno, Mezzanotte) on COMFORT to Judy;
```

## Revoca dei privilegi di oggetto

Se i privilegi di oggetto possono essere concessi, possono anche essere annullati. Questa sintassi è simile a quella del comando `grant`:

```
revoke { privilegio oggetto | all [privileges]}
       [(colonna [, colonna]. . . )]
[. { privilegio oggetto | all [privileges]}
   [(colonna [, colonna]. . . )]]. . .
on oggetto
from {utente | ruolo} [.{utente | ruolo}]
[cascade constraints] [force];
```

`revoke all` rimuove qualsiasi privilegio elencato precedentemente, da `SELECT` a `INDEX`; la revoca di singoli privilegi lascia invece inalterati gli altri che sono stati concessi. L'opzione `with grant option` viene revocata insieme con il privilegio cui è associata.

Se un utente definisce vincoli di integrità referenziale sull'oggetto, Oracle li rimuove se i privilegi sull'oggetto vengono revocati utilizzando l'opzione cascade constraints. L'opzione force si applica ai tipi di dati definiti dall'utente (si consulti il Capitolo 31). L'opzione force revoca il privilegio EXECUTE sugli oggetti dei tipi di dati definiti dall'utente con dipendenze di tabella o di tipo; tutti gli oggetti dipendenti vengono contrassegnati come non validi e i dati nelle tabelle dipendenti non sono più accessibili fino a quando il privilegio necessario non viene concesso nuovamente.

## Sicurezza con User

L'accesso alle tabelle può essere concesso in modo specifico, tabella per tabella e vista per vista, a ogni utente. Esiste, tuttavia, un'altra tecnica che in alcuni casi semplifica questo processo. Si devono riprendere i record di controllo della biblioteca:

```
select Nome, SUBSTR(Titolo,1,30) from BIBLIOTECA_PRESTITO;
```

NOME	SUBSTR(TITOLO,1,30)
JED HOPKINS	INNUMERACY
GERHARDT KENTGEN	WONDERFUL LIFE
DORAH TALBOT	EITHER/OR
EMILY TALBOT	ANNE OF GREEN GABLES
PAT LAVAY	THE SHIPPING NEWS
ROLAND BRANDT	THE SHIPPING NEWS
ROLAND BRANDT	THE DISCOVERERS
ROLAND BRANDT	WEST WITH THE NIGHT
EMILY TALBOT	MIDNIGHT MAGIC
EMILY TALBOT	HARRY POTTER AND THE GOBLET OF
PAT LAVAY	THE MISMEASURE OF MAN
DORAH TALBOT	POLAR EXPRESS
DORAH TALBOT	GOOD DOG, CARL
GERHARDT KENTGEN	THE MISMEASURE OF MAN
FRED FULLER	JOHN ADAMS
FRED FULLER	TRUMAN
JED HOPKINS	TO KILL A MOCKINGBIRD
DORAH TALBOT	MY LEDGER
GERHARDT KENTGEN	MIDNIGHT MAGIC

Per consentire a ogni utente che prende un libro in prestito di accedere alla tabella, restringendo però l'accesso a una vista costituita soltanto dalla propria riga, si potrebbero creare 19 viste distinte, ciascuna con nomi differenti nella clausola where e si potrebbero effettuare concessioni separate di queste viste per ogni persona. In alternativa, si può creare una vista la cui clausola where contiene User, la pseudocolonna, in questo modo:

```
create view MIO_PRESTITO as
select * from BIBLIOTECA_PRESTITO
where SUBSTR(Nome,1,INSTR(Nome,' ')-1) = User;.
```

Il proprietario della tabella BIBLIOTECA\_PRESTITO può creare questa vista e concedere agli utenti il privilegio SELECT. Quindi, un utente potrebbe eseguire una query sulla tabella BIBLIOTECA\_PRESTITO attraverso la vista MIO\_PRESTITO. La clausola where troverà il nome dell'utente nella colonna Nome (si veda la clausola where), e restituirà solo i record relativi all'utente.

Nell'esempio seguente viene creato un nome utente Jed e gli viene concesso l'accesso alla vista MIO\_PRESTITO. Se si effettuano selezioni da questa vista, si otterranno solo i libri presi in prestito da Jed.

```
create user jed identified by hop;
User created.

grant CREATE SESSION to jed;
Grant succeeded.

grant select on Pratica.MIO_PRESTITO to jed;
Grant succeeded.

connect jed/hop
Connected.

select * from Pratica.MIO_PRESTITO;

NOME
-----
TITOLO
-----
DATAPREST DATAREST
-----
JED HOPKINS
INNUMERACY
01-JAN-02 22-JAN-02

JED HOPKINS
TO KILL A MOCKINGBIRD
15-FEB-02 01-MAR-02
```

Ogni volta che si effettua una query su MIO\_PRESTITO, si otterranno dei record che dipendono dai valori della pseudocolonna Nome, l'utente di SQLPLUS la cui vista viene sottoposta a query.

## Concessione dell'accesso al pubblico

Invece di concedere l'accesso a ogni lavoratore, il comando grant può essere generalizzato al pubblico:

```
grant select on MIO_PRESTITO to public;
```

Questo dà l'accesso a tutti, compresi gli utenti creati dopo che questa concessione è stata effettuata. Tuttavia, ogni utente continuerà ad accedere alla tabella utilizzando come prefisso il nome utente del proprietario. Per evitare questo, un dba potrebbe creare un *sinonimo pubblico* (un nome accessibile a tutti gli utenti che sostituisce Pratica.MIO\_PRESTITO).

```
create public synonym MIO_PRESTITO for Pratica.MIO_PRESTITO;
```

Da questo momento, chiunque può accedere a MIO\_PRESTITO senza utilizzare il prefisso del proprietario dello schema (in questo caso, Pratica). Questo approccio dà enorme flessibilità per la sicurezza. I lavoratori potrebbero vedere solo il proprio stipendio, per esempio, in una tabella che contiene gli stipendi di tutti. Tuttavia, se un utente crea una tabella o una vista con lo stesso nome di un sinonimo pubblico, qualsiasi istruzione SQL futura di questo utente agisce su questa nuova tabella o vista e non su quella dello stesso nome a cui ha accesso il pubblico.

**NOTA** *Tutti questi accessi, e tutti i tentativi di operazioni a livello di sistema, come le connessioni, potrebbero essere sottoposti ad audit. Per un'introduzione alle capacità di controllo della sicurezza di Oracle, si consulti la voce AUDIT nel Capitolo 42.*

### 19.3 Concessione di risorse limitate

Quando si concedono delle quote di risorse in un database Oracle, si utilizza il parametro quota del comando `create user` o `alter user`, come mostrato nel listato seguente. In questo esempio, a Bob viene concessa una quota di 100 MB nella tablespace UTENTI.

```
alter user Bob
quota 100M on UTENTI;
```

La quota di spazio di un utente può essere stabilita quando l'utente viene creato, attraverso il comando `create user`. Se si desidera eliminare i limiti nella quota di spazio di un utente, si può concedere all'utente il privilegio di sistema `UNLIMITED TABLESPACE`. Per ulteriori dettagli su questi comandi, è possibile consultare le voci relative ai comandi `create user` e `alter user` nel Capitolo 42.

Anche i profili possono essere utilizzati per imporre limiti di altre risorse, come la quantità di tempo CPU o il tempo di inattività che può essere necessario per le richieste di un utente a Oracle. Viene creato un profilo che descrive dettagliatamente questi limiti di risorse e poi viene assegnato a uno o più utenti. Per i dettagli completi, si consultino le voci dei comandi `create profile` e `alter user` nel Capitolo 42.

### 19.4 Opzioni avanzate

Oltre alle funzioni descritte in questo capitolo, Oracle supporta anche funzioni di sicurezza avanzate. La caratteristica Virtual Private Database (VPD) di Oracle offre un modello di sicurezza simile a quello descritto in precedenza nel paragrafo “Sicurezza con User”. VPD aggiunge automaticamente delle clausole `where` a tutti i comandi eseguiti da utenti differenti, dando così la possibilità di creare programmi di sicurezza complessi che vengono imposti all'interno del database. Con VPD è possibile creare criteri che consentono di avere dati tratti da più aree di attività nella stessa tabella mentre gli utenti che effettuano la query sulla tabella visualizzano solo i dati della propria area di attività. In VPD si può utilizzare una procedura per convalidare un ruolo invece di incorporarne uno in un'applicazione (`create role nomeruolo identified using nomeprocedura`).

Un'altra opzione di sicurezza, Oracle Label Security, consente di assegnare etichette di sicurezza alle righe. Per esempio, si potrebbe creare un sistema in cui i dirigenti possono visualizzare tutte le righe mentre i livelli più bassi all'interno di un'organizzazione possono vedere soltanto

un sottoinsieme di righe. L'etichetta di sicurezza di ogni riga viene messa a confronto con i privilegi di sicurezza dell'utente durante l'accesso ai dati. Oracle Label Security è una soluzione adeguata quando le regole di sicurezza a livello di riga vengono imposte al meglio attraverso raggruppamenti gerarchici di righe per utenti.

Per maggiori dettagli su VPD e Oracle Label Security, si consulti la guida *Oracle9i Security Products and Features* e la documentazione *Oracle9i Security Overview*.

Gli amministratori di database possono generare funzioni per gestire la complessità delle password; per una documentazione e un esempio, si veda il file utlpwdmg.sql nella directory /rdbms/admin nella directory home del software Oracle.

## Capitolo 20

# Modifica degli ambienti di Oracle

- 20.1 **Indici**
- 20.2 **Tablespace e struttura del database**
- 20.3 **Segmenti di rollback e undo gestiti a livello di sistema**
- 20.4 **Cluster**
- 20.5 **Sequenze**

Come nel Capitolo 19, anche in questo viene esaminato un sottinsieme di funzioni per l'amministrazione del database (DBA) utilizzate dalla maggior parte degli utenti Oracle, quindi non ristrette ai dba. Queste funzioni includono indici sulle tabelle, cluster, generatori di sequenze e allocazioni di spazio per tabelle e indici. Questo capitolo mostra come sono strutturati internamente i database Oracle, aiutando a capire come funzionano realmente alcune caratteristiche di Oracle e come sono intercorrelate. Inoltre, vengono presentate solo alcune opzioni dei comandi `create index`, `create tablespace` e `create cluster`. Le opzioni complete sono presentate nei Capitoli 40 e 42, più avanti in questo libro.

## 20.1 Indici

Un *indice* è un concetto semplice. In genere si tratta di un elenco di *parole chiave* accompagnate dalla *posizione* delle informazioni relative a un particolare argomento. Per trovare informazioni sugli indici, per esempio, occorre cercare la parola “indici” nell’indice analitico di questo libro: si trova così il numero di questa pagina. La parola “indici” è la chiave, mentre i numeri di pagina indicano dove localizzare la trattazione degli indici in questo libro. Questo è correlato all’idea di chiave primaria in Oracle, che è stata descritta nel Capitolo 2.

Anche se gli indici non sono strettamente necessari per eseguire Oracle, velocizzano il processo. Per esempio, anche se è possibile trovare informazioni sugli indici semplicemente sfogliando il libro finché non si trova questa pagina, questo processo è lento e richiede molto tempo. Poiché l’indice analitico alla fine del libro è in ordine alfabetico, si può raggiungere velocemente il punto appropriato nell’indice (senza leggere tutte le voci) nel quale si trova la voce “indici”. Se la parola cercata non è comune, questa soluzione è sicuramente più rapida che leggere il libro dall’inizio. Questi stessi principi si applicano agli indici di Oracle. Per esempio, se si cerca un libro in particolare, si può eseguire una query con una condizione di limitazione sulla colonna Titolo:

```
select * from BIBLIOTECA  
where Editore = 'SCHOLASTIC';
```

Se la tabella BIBLIOTECA non ha un indice sulla colonna Editore, Oracle dovrà leggere ogni riga della tabella fino a quando non trova tutti gli editori che corrispondono alla clausola where della query. Se la tabella è piccola, questo potrebbe non avere conseguenze sulle prestazioni. Quando le dimensioni della tabella aumentano, il tempo necessario per restituire all'utente tutte le righe corrispondenti può influire negativamente sulle prestazioni dell'applicazione e dei processi che essa supporta.

Per velocizzare il recupero dei dati, si può creare un indice sulla colonna Editore. Pertanto, quando si esegue la stessa query, Oracle cerca per prima cosa nell'indice, che è in ordine, trovando così molto più velocemente l'editore denominato "SCHOLASTIC" (Oracle non legge ogni voce, ma salta direttamente nelle vicinanze del nome, proprio come si fa quando si consulta l'indice di un libro). La voce dell'indice fornisce a Oracle la dislocazione esatta nella tabella (e sul disco) delle righe relative a quell'editore.

Sapendo questo, è chiaro che indicizzare una colonna importante (una che probabilmente sarà presente in una clausola where) velocizza la risposta di Oracle a una query. La creazione di un indice velocizza anche query in cui due tabelle vengono unite tramite join, se le colonne cui si fa riferimento (attraverso la clausola where) sono incluse nell'indice. Questi sono i concetti fondamentali per gli indici; il resto del capitolo mostra un certo numero di caratteristiche aggiuntive e alcuni problemi relativi agli indici che influiscono sulla loro velocità di funzionamento. Per informazioni sull'influsso che gli indici esercitano sull'ottimizzazione delle query, si rimanda al Capitolo 38.

## Creazione di un indice

La creazione di un indice avviene attraverso il comando `create index`. Quando si definisce una chiave primaria o una colonna unica durante le fasi di creazione o gestione di una tabella, Oracle crea automaticamente un indice unico per supportare questo vincolo. La sintassi completa del comando `create index` è mostrata nel Capitolo 42. La forma più utilizzata di questo comando è la seguente:

```
create [bitmap | unique] index indice on tabella(colonna [.colonna] . . .) [reverse];
```

dove *indice* dev'essere un nome unico e seguire le convenzioni di denominazione per le colonne in Oracle. *Tabella* è semplicemente il nome della tabella su cui viene costruito l'indice e *colonna* è il nome della colonna.

Gli indici bitmap consentono di creare indici utili sulle colonne con pochissimi valori distinti; per questo occorre leggere il paragrafo "Creazione di indici bitmap" più avanti in questo capitolo. La parola chiave *reverse* indica a Oracle di invertire i byte del valore indicizzato, operazione che potrebbe migliorare la distribuzione dei dati e dei processi di elaborazione durante l'inserimento di più valori di dati sequenziali.

È possibile costruire un indice singolo su più colonne elencando le colonne una di seguito all'altra, separate da virgolette. Per gli esempi seguenti, la tabella BIBLIOTECA\_AUTORE è stata creata senza una chiave primaria:

```
create table BIBLIOTECA_AUTORE
(Titolo      VARCHAR2(100),
NomeAutore  VARCHAR2(50),
constraint TitoloFK Foreign key (Titolo) references BIBLIOTECA(Titolo),
constraint NomeAutoreFK Foreign key (NomeAutore)
references AUTORE(NomeAutore));
```

## Imposizione dell'unicità

Nel Capitolo 2 è stato spiegato che una serie di tabelle è detta in terza forma normale se tutte le colonne in ogni riga della tabella dipendono soltanto dalla chiave primaria. Nella tabella BIBLIOTECA\_AUTORE la chiave primaria è la combinazione delle colonne Titolo e NomeAutore. In altre tabelle, la chiave primaria può essere un identificativo dei dipendenti, un identificativo dei clienti, un numero di conto o, in una banca, una combinazione di un numero di filiale e uno di conto.

In ognuno di questi casi, l'unicità della chiave primaria è fondamentale. In una banca con numeri di conto duplicati o in un sistema di fatturazione con identificativi di clienti duplicati si avrebbero gravi problemi, poiché le transazioni verrebbero assegnate a conti appartenenti a persone diverse, ma che hanno la stessa chiave primaria (questo è il motivo per cui i nomi di solito non sono utilizzati come chiavi primarie: ci sono troppi doppioni). Per evitare questo pericolo, il database aiuta a impedire la creazione di chiavi primarie duplicate. Oracle offre due soluzioni:

- Si può garantire l'unicità di una chiave attraverso l'inserimento in indici o vincoli.
- Si possono utilizzare i generatori di sequenze (trattati più avanti in questo capitolo).

### Creazione di un indice unico

Esistono tre modi per creare un indice unico sulla combinazione delle colonne Titolo e NomeAutore nella tabella BIBLIOTECA\_AUTORE: creando un vincolo di chiave primaria, creando un vincolo unico oppure creando un indice unico. Se si crea un vincolo, si potranno generare anche delle chiavi primarie che si riferiscono proprio a questo vincolo. Se si crea per primo l'indice unico, sarà sempre possibile creare una chiave primaria sulla tabella: Oracle userà l'indice già esistente come indice di chiave primaria.

Il listato seguente mostra il comando `create index` per questo indice a più colonne:

```
create unique index BA$TITOLO_AUTORE
  on BIBLIOTECA_AUTORE(Titolo, NomeAutore);
```

Il metodo della chiave primaria è mostrato di seguito:

```
alter table BIBLIOTECA_AUTORE
  add constraint BA_PK primary key (Titolo, NomeAutore);
```

Per creare un vincolo unico, è sufficiente sostituire la clausola `create primary key` con `unique`. Quando si cerca di creare un indice unico su una tabella che contiene già dei dati, il comando non riuscirà se esistono dei duplicati. Se l'istruzione `create unique index` riesce, qualsiasi tentativo futuro di inserire (o aggiornare) una riga che crea un duplicato nella chiave fallisce e si ottiene come risultato questo messaggio di errore:

```
ERROR at line 1: ORA-00001: unique constraint (BIBLIOTECA_AUTORE.BA_PK) violated
```

Quando si crea un indice, questo richiede dello spazio di memorizzazione. Per dettagli che riguardano la posizione degli indici creati, si rimanda al paragrafo “Posizionamento di un indice nel database” più avanti in questo capitolo.

### Creazione di un indice bitmap

Per facilitare la messa a punto di query che utilizzano colonne non selettive nelle condizioni limitative, si possono utilizzare indici bitmap. Questi indici vanno utilizzati solo se i dati sono

aggiornati poco frequentemente, in quanto accrescono il costo delle transazioni di manipolazione dei dati sulle tabelle che indicizzano.

Gli indici bitmap sono appropriati quando vengono impiegate colonne non selettive come condizioni limitative in una query. Per esempio, se ci sono pochissimi valori Valutazione distinti in una tabella BIBLIOTECA molto grande, in genere non è possibile creare un indice B-tree tradizionale su Valutazione, anche se questa colonna è utilizzata con frequenza nella clausola where. Tuttavia, Valutazione potrebbe essere in grado di trarre vantaggio da un indice bitmap.

Internamente, un indice bitmap associa i valori distinti delle colonne a ogni record. Per questo esempio, si supponga che esistono cinque valori di Valutazione (1, 2, 3, 4 e 5) in una tabella BIBLIOTECA di grandi dimensioni. Dato che esistono cinque valori di Valutazione, ci saranno anche cinque voci bitmap distinte per l'indice bitmap di Valutazione. Se le prime cinque righe nella tabella hanno un valore di Valutazione pari a 1, e le cinque righe successive un valore pari a 2, le voci bitmap di Valutazione sono come quelle mostrate nel listato seguente:

Rating bitmaps:

```
1: < 1 1 1 1 1 0 0 0 0 0 >
2: < 0 0 0 0 0 1 1 1 1 1 >
3: < 0 0 0 0 0 0 0 0 0 0 >
4: < 0 0 0 0 0 0 0 0 0 0 >
5: < 0 0 0 0 0 0 0 0 0 0 >
```

Nel listato precedente ciascuna colonna di zeri e di uno rappresenta una riga nella tabella BIBLIOTECA. Dato che vengono considerate dieci righe, vengono visualizzati altrettanti valori bitmap. Leggendo la bitmap per Valutazione, i primi cinque record hanno un valore di Valutazione pari a 1 (i valori "1"), e i cinque successivi no (i valori '0'). Esiste una voce bitmap separata per ogni valore possibile.

L'ottimizzatore di Oracle è in grado di convertire dinamicamente voci di indici bitmap in RowID durante l'elaborazione delle query. Questa capacità di conversione consente all'ottimizzatore di utilizzare indici sia su colonne che hanno molti valori distinti (attraverso indici B-tree), sia su quelle che hanno pochi valori distinti (attraverso indici bitmap).

Per creare un indice bitmap, occorre utilizzare la clausola bitmap del comando create index, come mostrato nel listato seguente. Conviene indicare che si tratta di un indice bitmap nel nome, in modo che possa essere rilevato facilmente durante le operazioni di messa a punto.

```
create bitmap index BIBLIOTECA$BITMAP_VALUTAZIONE
    on BIBLIOTECA(Valutazione);
```

Se si decide di utilizzare indici bitmap, è necessario considerare i vantaggi nelle prestazioni durante l'esecuzione delle query rispetto agli svantaggi durante i comandi di manipolazione dei dati. Più indici bitmap sono presenti su una tabella, maggiore è il costo nel corso di ogni transazione. Non conviene utilizzare indici bitmap su una colonna cui vengono aggiunti frequentemente nuovi valori. Ogni aggiunta di un nuovo valore alla colonna Valutazione richiede la creazione di una nuova bitmap corrispondente.

## Quando creare un indice

Gli indici sono particolarmente utili su tabelle molto grandi, su colonne che probabilmente saranno presenti in clausole where come una semplice uguaglianza, come la seguente:

```
where NomeAutore = 'STEPHEN JAY GOULD'
    and Valutazione = '5'
```

oppure in join, come questo:

```
where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo
```

Per ulteriori dettagli sull'impiego che l'ottimizzatore fa degli indici, si rimanda al Capitolo 38. Se non vi è alcuna clausola where, non viene utilizzato alcun indice.

## Varietà nelle colonne indicizzate

Gli indici (B-tree) tradizionali sono utili soprattutto su colonne con una significativa varietà di dati. Per esempio, una colonna che indica se un'azienda attualmente è tra i clienti con una S o una N non va molto bene per un indice tradizionale e potrebbe effettivamente rallentare una query; un indice bitmap sarebbe una scelta assai migliore per un esempio di questo tipo. Una colonna con numeri telefonici è un buon candidato per un indice B-tree. Una colonna con i prefissi telefonici sarebbe di poca importanza, poiché dipende dalla distribuzione di valori di prefissi unici nella tabella.

In un indice a più colonne, per prima cosa occorre assicurarsi di collocare la colonna in una posizione che garantisca un accesso frequente. A partire da Oracle9i è possibile sfruttare la caratteristica skip-scan dell'ottimizzatore per utilizzare indici a più colonne anche se la prima colonna non è citata nella query. Per esempi sull'uso degli indici, si rimanda al Capitolo 38.

Conviene lasciare le piccole tabelle senza indici, eccetto che per imporre l'unicità della chiave primaria. Si dice che una tabella è piccola quando accetta meno di cinque blocchi di database; oltre questo valore, l'indicizzazione sarà quasi sempre produttiva.

Gli indici bitmap offrono un'alternativa di indicizzazione attuabile per colonne che hanno pochissimi valori distinti. Gli indici bitmap vengono comunemente utilizzati per colonne di "flag" che sono limitate a valori come 'S' e 'N'. Gli indici bitmap sono particolarmente efficaci quando in una query ne sono utilizzati diversi; l'ottimizzatore può valutare velocemente le bitmap e determinare quali righe soddisfano tutti i criteri per cui gli indici bitmap sono disponibili.

## Quanti indici utilizzare su una tabella

Su una singola tabella si possono creare molti indici, ciascuno dei quali contiene diverse colonne. Lo svantaggio provocato dall'indicizzazione di troppe colonne risulta evidente nella velocità di inserimento di nuove righe: ogni indice deve avere anche una nuova voce quando viene effettuato un inserimento. Se la tabella è utilizzata soprattutto per query, il solo costo dell'includere in un indice più colonne possibili (che naturalmente hanno una certa varietà e sono destinate a essere utilizzate nella clausola where) è l'utilizzo di più spazio sul disco. A rigore, da un punto di vista delle prestazioni relative al caricamento dei dati, è meglio avere pochi indici con molte colonne che avere molti indici con poche colonne.

Con l'eccezione degli indici cluster (trattati più avanti in questo capitolo), le colonne che sono NULL non compaiono in un indice. Gli indici basati su più di una colonna hanno una voce se almeno una delle colonne non è NULL. Se per una data riga tutte le colonne sono NULL, nell'indice non compare alcuna voce.

## Posizionamento di un indice nel database

È possibile specificare dove dev'essere posto un indice per una tabella, assegnando una particolare tablespace. Come è stato brevemente accennato nel Capitolo 19, una *tablespace* è una sezione

ne logica del database, che corrisponde a uno o più file di dati. I file di dati forniscono la memoria fisica per il database, sezioni del disco dove sono memorizzati gli indici e le tabelle. I database possono avere molte tablespace, ciascuna con un proprio nome. Per ridurre i potenziali conflitti di disco, un indice di una tabella dovrebbe essere posto in una tablespace che si trova su un disco fisicamente separato dalla sua tabella corrispondente.

Per specificare la tablespace in cui collocare un indice, occorre utilizzare la normale istruzione `create index` seguita dalla parola `tablespace` e dal nome della tablespace, come mostrato di seguito:

```
create unique index BA$TITOLO_AUTORE
    on BIBLIOTECA_AUTORE(Titolo, NomeAutore)
tablespace BA_INDICI;
```

In questo esempio, `BA_INDICI` è il nome dato alla tablespace creata in precedenza dall'amministratore del database. L'uso dell'opzione `tablespace` in un'istruzione `create index` consente di separare fisicamente le tabelle dagli indici a esse associati.

Quando si crea una chiave primaria o un vincolo unico, Oracle crea automaticamente un indice per imporre l'unicità. A meno che non si specifichi diversamente, l'indice viene creato nella stessa tablespace della tabella cui si applica il vincolo, e utilizza i parametri per la memorizzazione predefiniti per quella tablespace. Dal momento che la locazione dell'area di memorizzazione è spesso inadeguata, conviene utilizzare la clausola `using index` quando si creano chiavi primarie e vincoli unici.

La clausola `using index` consente di specificare i parametri di memorizzazione e la locazione della tablespace per un indice creato da un vincolo. Nell'esempio seguente, viene creata una chiave primaria sulla tabella `BIBLIOTECA_AUTORE` e viene collocata nella tablespace `BA_INDICI`. Questo esempio presuppone che nelle colonne specificate non esista alcun indice.

```
alter table BIBLIOTECA_AUTORE
    add constraint BA_PK primary key (Titolo, NomeAutore)
    using index tablespace BA_INDICI;
```

Per ulteriori opzioni relative alla clausola `using index` si può consultare la voce “*Vincoli di integrità*” nel Capitolo 42. Per le opzioni relative alle prestazioni legate alla creazione di indici si può consultare la voce dedicato a `create index` sempre nel Capitolo 42.

## Ricostruzione di un indice

Oracle mette a disposizione una funzionalità per la *ricostruzione rapida degli indici* che consente di ricreare un indice senza dover rimuovere quello già esistente. L'indice attualmente disponibile è utilizzato come origine di dati per il nuovo indice, al posto della tabella. Durante la ricostruzione dell'indice è possibile cambiarne i parametri di memorizzazione e l'assegnamento della tablespace.

Nell'esempio seguente viene ricostruito l'indice `BA_PK` (attraverso la clausola `rebuild`). I suoi parametri di memorizzazione vengono modificati in modo da utilizzare una dimensione di estensione iniziale pari a 8MB e una dimensione di estensione successiva di 4MB, nella tablespace `BA_INDICI`.

```
alter index BA_PK rebuild
storage (initial 8M next 4M pctincrease 0)
tablespace BA_INDICI;
```

**NOTA** Quando l'indice BA\_PK viene ricostruito, ci dev'essere spazio sufficiente per entrambi gli indici, quello vecchio e quello nuovo. Dopo che il nuovo indice è stato creato, quello vecchio viene rimosso e il suo spazio liberato.

Quando si crea un indice basato su colonne precedentemente indicizzate, Oracle può essere in grado di utilizzare gli indici già esistenti come origine di dati per quello nuovo. Per esempio, se si crea un indice a due colonne sulle colonne Titolo e NomeAutore, e in seguito si decide di creare un indice soltanto sulla colonna Titolo, Oracle utilizza l'indice già esistente come origine di dati per il nuovo indice. Come risultato, le prestazioni dei comandi create index migliorano, se si creano gli indici in modo che possano sfruttare questa caratteristica.

A partire da Oracle8i, è possibile ricostruire gli indici mentre vi si accede tramite la clausola rebuild online del comando alter index.

## Indici basati su funzioni

A partire da Oracle8i, esiste la possibilità di creare *indici basati su funzioni*. Prima di Oracle8i, qualsiasi query eseguisse una funzione su una colonna non poteva servirsi dell'indice della colonna. Pertanto, questa query non può usare un indice sulla colonna Titolo:

```
'MIO REGISTRO';
```

mentre questa query sì:

```
select * from BIBLIOTECA
  where Titolo = 'MIO REGISTRO';
```

poiché non esegue la funzione UPPER sulla colonna Titolo. A partire da Oracle8i, si possono creare indici che permettono agli accessi tramite indice di supportare accessi basati sulle funzioni. Invece di creare un indice sulla colonna Titolo, si può creare un indice sull'espressione della colonna SUPERIORE(Titolo), come mostrato nel listato seguente:

```
create index BIBLIOTECA$SUPERIORE_TITOLO on
BIBLIOTECA(UPPER(Titolo));
```

Anche se gli indici basati sulle funzioni possono essere utili, è opportuno considerare queste domande quando si creano indici di questo tipo:

- Esiste la possibilità di limitare le funzioni che verranno utilizzate sulla colonna? In caso affermativo, si può impedire che tutte le funzioni vengano eseguite sulla colonna?
- Si ha spazio di memoria sufficiente per altri indici?
- Quando si rimuove la tabella, con essa vengono rimossi più indici (e di conseguenza più extent) di prima; in che modo questo influisce sul tempo necessario per rimuovere la tabella?

Gli indici basati su funzioni sono utili, tuttavia andrebbero implementati con parsimonia. Più indici si creano su una tabella, più lente saranno le operazioni di inserimento, aggiornamento e cancellazione.

**NOTA** Il DBA deve attivare le funzionalità di "riscrittura delle query" nel database per consentire l'uso degli indici basati su funzioni. Il parametro di inizializzazione QUERY\_READONLY\_ENABLED è impostato a FALSE per default.

## 20.2 Tablespace e struttura del database

Tutti conoscono il concetto di file; si tratta di un'area sul disco dotata di un nome in cui vengono memorizzate le informazioni. La sua dimensione in genere non è fissa: se si aggiungono delle informazioni al file, questo può ingrandirsi e occupare più spazio sul disco, fino al massimo disponibile. Questo processo è gestito dal sistema operativo, e spesso implica la distribuzione delle informazioni nel file su varie sezioni più piccole del disco che non sono fisicamente vicine le une alle altre. Il sistema operativo si occupa della connessione logica di queste sezioni più piccole senza che l'utente se ne renda conto: per lui il file sembra sempre un'unità intera.

Oracle utilizza i file come parti integranti del suo schema organizzativo, tuttavia la sua struttura va al di là del concetto di file. Un file di dati è un file del sistema operativo utilizzato per memorizzare i dati di Oracle. Ogni file di dati è assegnato a una tablespace, ossia una divisione logica all'interno del database. Solitamente, le tablespaces includono la tablespace SYSTEM (per il dizionario di dati interno di Oracle), USERS (per gli oggetti degli utenti) e altre ancora per le tabelle delle applicazioni, per gli indici e per strutture di database aggiuntive.

I file di dati hanno una dimensione fissa, oppure possono ingrandirsi automaticamente quando si riempiono, fino a raggiungere un limite ben definito. Per aggiungere altro spazio a una tablespace, è possibile ingrandire manualmente i file di dati oppure aggiungerne di nuovi. Quindi, le nuove righe possono essere aggiunte alle tabelle già esistenti, che possono avere righe in più file di dati.

Ogni tabella ha una singola area di spazio su disco, chiamata *segmento*, e ad essa riservata nella tablespace. Ogni segmento a sua volta ha un'area iniziale di spazio su disco, chiamata *extent iniziale*, a esso riservata nella tablespace. Quando il segmento ha utilizzato tutto il suo spazio, gli viene assegnato l'*extent successivo*, ovvero un'altra singola area di spazio su disco. Quando ha utilizzato anche questo, gliene viene assegnato un'altro. Il processo continua con ogni tabella finché l'intera tablespace non è piena. A questo punto, occorre aggiungere un nuovo file alla tablespace o estenderne i file prima di poter ampliare le tabelle.

Ogni database contiene anche una tablespace di sistema, la quale contiene il dizionario di dati, i nomi e le posizioni di tutte le tablespaces, le tabelle, gli indici e i cluster per il database. Gli oggetti all'interno della tablespace SYSTEM sono posseduti dagli utenti SYS e SYSTEM; nessun altro utente dovrebbe possedere oggetti in questa tablespace, in quanto potrebbero influire sul resto del database.

### create tablespace

Il comando `create tablespace` consente di assegnare immediatamente alla tablespace uno o più file. Inoltre, specifica una storage clause predefinita per ogni tabella creata, senza che nell'istruzione `create table` o `create index` venga menzionata una storage clause esplicita. Per creare le tablespaces occorre disporre dei privilegi di DBA; per un'analisi della creazione e della gestione delle tablespaces si rimanda al Capitolo 40.

I valori predefiniti per la memorizzazione dipendono dal sistema operativo. I valori minimo e massimo per ciascuna di queste opzioni sono disponibili nel Capitolo 42 alle voci `create table` e "Memoria." Queste opzioni possono essere modificate con il comando `alter tablespace`. Il comando `create table` per la tabella BIBLIOTECA, inserita nella tablespace PRATICA, ha l'aspetto seguente:

```
create table BIBLIOTECA
(Titolo      VARCHAR2(100) primary key,
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
```

```

Valutazione      VARCHAR2(2),
constraint CATFK foreign key (NomeCategoria)
references CATEGORIA(NomeCategoria)
)
tablespace PRATICA
;

```

In questa forma, la tabella BIBLIOTECA eredita le definizioni di memorizzazione predefinite della tablespace PRATICA. Per ridefinire questi valori predefiniti, occorre utilizzare la clausola storage nel comando create table:

```

create table BIBLIOTECA
(Titolo      VARCHAR2(100) primary key,
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Valutazione  VARCHAR2(2),
constraint CATFK foreign key (NomeCategoria)
references CATEGORIA(NomeCategoria)
)
tablespace PRATICA
storage (initial 4M next 4M minextents 2 pctincrease 0)
;

```

Se si usano delle tabelle temporanee (si veda il Capitolo 13), è possibile creare una tablespace appositamente per le loro esigenze di memorizzazione. Per supportare questo tipo particolare di tabella, si usa il comando create temporary tablespace (descritto in dettaglio nel Capitolo 42).

A partire da Oracle8i esiste la possibilità di creare tablespace gestite a livello locale; con Oracle9i questo tipo di tablespace sarà quello predefinito. In una tablespace gestita a livello locale, Oracle mantiene un bitmap nelle intestazioni dei file di dati per gestire la locazione dei dati all'interno degli stessi file di dati. Se invece non si usa una tablespace gestita a livello locale, le informazioni sulla gestione degli extent sono mantenute dalle tabelle nel dizionario di dati (una tablespace gestita a livello di dizionario). In linea di massima, le tablespace gestite a livello locale offrono un modo più semplice e più coerente per gestire lo spazio all'interno delle tablespace. Nel Capitolo 40 viene descritto dettagliatamente l'argomento relativo alla gestione dello spazio.

I DBA possono modificare una tablespace attraverso il comando alter tablespace. È possibile modificare le tablespace in modo che siano di sola lettura:

```
alter tablespace PRATICA read only;
```

oppure trasformarle da sola lettura in tablespace scrivibili:

```
alter tablespace PRATICA read write;
```

I dati contenuti in una tablespace di sola lettura non possono essere alterati. A partire da Oracle8i, le tablespace possono essere spostate tra i vari database; questo concetto, che prende il nome di tablespace trasportabile, richiede che le prime tablespace a essere spostate vengano collocate in uno stato di sola lettura.

## **Tablespace temporanee**

Quando si esegue un comando che effettua un'operazione di ordinamento o raggruppamento, Oracle può creare un segmento temporaneo per gestire i dati. Questo segmento temporaneo

viene creato in una tablespace temporanea, e l'utente che esegue il comando non deve preoccuparsi di gestire anche i dati. Oracle crea il segmento temporaneo in modo dinamico e rilascia il suo spazio dopo che l'operazione è stata completata. Se non c'è spazio temporaneo sufficiente e i file di dati della tablespace temporanea non sono in grado di allargarsi da soli, il comando fallisce. A ciascun utente del database è associata una tablespace temporanea, ma potrebbe esserne solo una condivisa da tutti gli utenti. A partire da Oracle9i, i DBA potranno definire una tablespace temporanea predefinita per tutti gli utenti.

### 20.3 Segmenti di rollback e undo gestiti a livello di sistema

Con il comando rollback di SQL gli utenti possono annullare le transazioni che sono state effettuate su un database. Il comando rollback è disponibile per qualsiasi transazione di inserimento, aggiornamento o cancellazione; non è disponibile per le modifiche agli oggetti del database (come i comandi alter table). Quando si selezionano dati che un altro utente sta modificando, Oracle ricorre ai segmenti di rollback per mostrare com'erano i dati prima delle modifiche.

#### Uso dei segmenti di rollback da parte del database

I segmenti di rollback sono coinvolti in ogni transazione che avviene all'interno del database. Il numero e la dimensione dei segmenti di rollback disponibili sono specificati dal DBA durante la creazione del database (si consulti la voce *create rollback segment* nel Capitolo 42). Dato che i segmenti di rollback vengono creati nelle tablespace, la loro dimensione massima è limitata alla quantità di spazio disponibile nei file di dati della tablespace.

Il database assegna ai segmenti di rollback le varie transazioni seguendo una modalità circolare che parte da quella eseguita meno di recente, il che comporta una distribuzione abbastanza uniforme del numero di transazioni tra il gruppo di segmenti di rollback. Anche se è possibile specificare il segmento di rollback che dovrebbe utilizzare una transazione, la maggior parte delle transazioni usa quello predefinito. A causa del metodo di assegnazione circolare, solitamente non conviene avere segmenti di rollback con dimensioni diverse.

Se si usano i segmenti di rollback, sarebbe opportuno lavorare con il proprio DBA per capire qualsiasi variazione di dimensione si verifichi. Alcuni database possono mettere a disposizione segmenti di rollback molto grandi per transazioni di grandi dimensioni; le stesse transazioni non riuscirebbero se venissero eseguite con segmenti di rollback più piccoli.

Se la transazione supera la quantità di spazio disponibile nel proprio segmento di rollback, e questo non può ingrandirsi ulteriormente, la transazione fallisce. Se si esegue una query che richiede molto tempo, e i dati su cui si basa sono cambiati e la versione precedente di questi dati non è più disponibile nei segmenti di rollback, la query non riuscirà. È molto difficile per gli sviluppatori di applicazioni capire in che modo i DBA hanno configurato i segmenti di rollback e come le loro applicazioni modificano i dati.

#### Uso di tablespace di undo

A partire da Oracle9i è possibile ricorrere alla *gestione automatica degli undo* per collocare tutti i dati di undo in un'unica tablespace. Quando il DBA crea una tablespace di undo, Oracle gestisce la memorizzazione, la conservazione e l'utilizzo dello spazio per i dati rollback grazie agli

SMU (undo gestiti a livello di sistema, system-managed undo). Quando si imposta un tempo di conservazione (nel file dei parametri di inizializzazione del database), Oracle farà del suo meglio per conservare tutti i dati di undo eseguiti nel database per il numero di secondi specificato. Con questa impostazione, qualsiasi query duri meno del tempo di conservazione non dovrebbe provocare un errore, purché la tablespace di undo sia stata dimensionata in modo corretto. Durante l'esecuzione del database, i DBA possono modificare il valore del parametro UNDO\_RETENTION con il comando alter system.

## 20.4 Cluster

La creazione di cluster è un metodo per memorizzare tabelle che sono strettamente correlate, e spesso unite tramite join nella stessa area sul disco. Per esempio, invece di porre la tabella BIBLIOTECA in una sezione del disco e la tabella BIBLIOTECA\_AUTORE in un'altra, si potrebbero inserire le loro righe in una stessa area, chiamata *cluster*. La *chiave di cluster* è la colonna, o le colonne, con cui le tabelle sono in genere unite tramite join in una query (per esempio Titolo per le tabelle BIBLIOTECA e BIBLIOTECA\_AUTORE). Si possono unire in un cluster solo tabelle di cui si è proprietari.

La sintassi di base del comando create cluster è la seguente:

```
create cluster cluster (colonna tipodati [.colonna
tipodati]. . .) [altre opzioni];
```

Il nome del cluster (*cluster*) segue le convenzioni di denominazione delle tabelle, mentre *colonna* e *tipodati* sono rispettivamente il nome e il tipo di dati della colonna che si vuole utilizzare come chiave di cluster. Il nome della colonna (*colonna*) può coincidere con quello di una delle colonne di una tabella che si desidera inserire nel cluster, oppure può essere qualsiasi altro nome valido. Ecco un esempio:

```
create cluster LIBROeAUTORE (Col1 VARCHAR2(100));
```

Cluster created.

Questo codice crea un cluster (uno spazio riservato, come avverrebbe per una tabella) che non contiene nulla. L'uso di Col1 come nome per la chiave di cluster è irrilevante; infatti, non verrà più utilizzato. Tuttavia, la sua definizione dovrebbe corrispondere a quella della chiave primaria della tabella da aggiungere. Successivamente, vengono create delle tabelle da includere nel cluster:

```
create table BIBLIOTECA
(Titolo      VARCHAR2(100) primary key,
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Valutazione  VARCHAR2(2),
constraint CATFK foreign key (NomeCategoria)
references CATEGORIA(NomeCategoria)
)
cluster LIBROeAUTORE (Titolo);
```

Prima di inserire delle righe in BIBLIOTECA, è necessario creare un indice del cluster:

---

```
create index LIBROeAUTOREndx
on cluster LIBROeAUTORE;
```

Si ricordi che la presenza della clausola `cluster` preclude l'utilizzo delle clausole `tablespace` o `storage`. È opportuno osservare come questa struttura sia diversa dall'istruzione `create table` standard:

```
create table BIBLIOTECA
(Titolo      VARCHAR2(100) primary key,
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Valutazione  VARCHAR2(2),
constraint CATFK foreign key (NomeCategoria)
references CATEGORIA(NomeCategoria)
);
```

Nella prima istruzione `create table`, la clausola `cluster LIBROeAUTORE` (`Titolo`) *segue* la parentesi di chiusura che racchiude l'elenco di colonne create nella tabella. `LIBROeAUTORE` è il nome del cluster creato precedentemente. `Titolo` è la colonna di questa tabella che viene memorizzata nella chiave di cluster `Col1`. È possibile avere più chiavi di cluster nell'istruzione `create cluster` e più colonne memorizzate in queste chiavi nell'istruzione `create table`. Occorre notare che in nessun punto un'istruzione specifica esplicitamente che la colonna `Titolo` dovrà essere memorizzata nella chiave di cluster `Col1`. L'accoppiamento viene realizzato solo in base alla posizione: `Col1` e `Titolo` erano entrambi i primi oggetti menzionati nelle rispettive istruzioni cluster. Colonne e chiavi di cluster multiple sono accoppiate la prima con la prima, la seconda con la seconda, la terza con la terza e così via. Ora, viene aggiunta al cluster una seconda tabella:

```
create table BIBLIOTECA_AUTORE
(Titolo      VARCHAR2(100),
NomeAutore   VARCHAR2(50),
constraint TitleFK Foreign key (Titolo) references BIBLIOTECA(Titolo),
constraint NomeAutoreFK Foreign key (NomeAutore)
references AUTORE(NomeAutore)
)
cluster LIBROeAUTORE (Titolo);
;
```

Quando queste due tabelle vengono unite in un cluster, ogni singolo `Titolo` viene memorizzato solo una volta nella chiave di cluster. A ogni `Titolo` vengono associate le colonne di entrambe le tabelle.

I dati di queste tabelle sono effettivamente memorizzati in una singola locazione, come se il cluster fosse una grande tabella che contiene dati ricavati da entrambe le tabelle che lo costituiscono.

Un altro tipo di cluster, detto *hash cluster*, utilizza i valori delle colonne di cluster per determinare la locazione fisica in cui è memorizzata la riga. Per maggiori informazioni si rimanda alla voce `create cluster` nel Capitolo 42.

## 20.5 Sequenze

Con una sequenza è possibile assegnare numeri unici, come identificativi dei clienti, alle colonne del database; non è necessario creare una tabella e del codice particolare per tenere traccia dei

numeri unici utilizzati. Questo si ottiene utilizzando il comando `create table`, come mostrato di seguito:

```
create sequence IDCliente increment by 1 start with 1000;
```

Questo codice crea una sequenza cui si può accedere durante l'esecuzione dei comandi `insert` e `update` (anche `select`, ma è alquanto raro). Solitamente, il valore di sequenza unico viene creato con un'istruzione simile a quella seguente:

```
insert into CLIENTE
  (Nome, Contatto, ID)
values
  ('COLE CONSTRUCTION', 'VERONICA',IDCliente.NextVal);
```

`NextVal` associato a `IDCliente` comunica a Oracle che si desidera il numero della prossima sequenza disponibile dalla sequenza `IDCliente`. Questo numero è sicuramente unico; Oracle non lo assegna a nessun altro. Per usare lo stesso numero più di una volta (come in una serie di inserimenti in tabelle correlate), `CurrVal` viene impiegato al posto di `NextVal`, *dopo il primo utilizzo*. In altre parole, l'uso di `NextVal` garantisce che la tabella della sequenza venga incrementata e che il numero ottenuto sia unico, quindi la prima volta si deve usare `NextVal`. Dopo aver utilizzato `NextVal`, quel numero viene memorizzato in `CurrVal` per impiegarlo altrove, finché non si utilizza di nuovo `NextVal`, e a quel punto sia `NextVal` sia `CurrVal` cambiano, assumendo il valore del nuovo numero della sequenza.

Se si utilizza `NextVal` e `CurrVal` in una singola istruzione SQL, entrambi contengono il valore recuperato da `NextVal`. Nessuno di questi può essere impiegato in una subquery, come colonna nella clausola `select` di una vista, con `DISTINCT`, `UNION`, `INTERSECT` o `MINUS` e nelle clausole `order by`, `group by` o `having` di un'istruzione `select`.

Si possono anche memorizzare i valori delle sequenze nella cache per un accesso più veloce, ed è possibile far ripartire il ciclo della sequenza dal valore iniziale quando viene raggiunto il valore massimo. Per maggiori informazioni si rimanda alla voce `create sequence` nel Capitolo 42.



## Capitolo 21

# Uso di SQL\*Loader per caricare i dati

- 21.1 **Il control file**
- 21.2 **Avvio del caricamento**
- 21.3 **Note sulla sintassi del control file**
- 21.4 **Gestione delle operazioni  
di caricamento dei dati**
- 21.5 **Messa a punto delle operazioni  
di caricamento dei dati**
- 21.6 **Altri miglioramenti di Oracle9i**

**N**e gli script memorizzati sul CD-ROM allegato al libro viene eseguito un grande numero di comandi `insert`. Al posto di questi inserimenti, si potrebbe creare un file contenente i dati da caricare, quindi usare l'utility SQL\*Loader di Oracle per caricarli. Questo capitolo offre una panoramica sull'impiego di SQL\*Loader e sulle sue capacità principali. Nel Capitolo 40, verranno analizzate altre due utility per lo spostamento di dati, Export e Import. SQL\*Loader, Export e Import sono descritte in dettaglio in *Oracle9i Database Utilities*, fornito con la documentazione standard di Oracle.

SQL\*Loader carica i dati da file esterni in tabelle contenute nel database Oracle. SQL\*Loader richiede due file principali: un file di dati, che contiene le informazioni da caricare, e un control file che contiene informazioni relative al formato dei dati, i record e i campi contenuti nel file, l'ordine in cui andranno caricati e, se necessario, i nomi dei file che verranno utilizzati per i dati. Inoltre, si possono unire le informazioni del control file con quelle del file di dati, sebbene questi due siano in genere separati per semplificare il riutilizzo del control file.

Una volta eseguito, SQL\*Loader creerà automaticamente un file di log e un file “non valido”. Il file di log registra lo stato del carico, come il numero di righe elaborate e il numero di righe eseguite; il file “non valido” conterrà tutte le righe rifiutate durante il caricamento per errori nei dati, come valori non univoci nelle colonne di chiave primaria.

All'interno del control file è possibile specificare altri comandi allo scopo di regolare i criteri di carico. Se una riga non soddisfa questi criteri, verrà scritta in un file “di scarto”. I file di log, “non valido” e di scarto avranno rispettivamente le estensioni `.log`, `.bad` e `.dsc`. Solitamente, i control file presentano l'estensione `.ctl`.

SQL\*Loader è un'utility che per diversi motivi si rivela molto potente per il caricamento dei dati.

- È estremamente flessibile, tanto da consentire la manipolazione dei dati mentre vengono caricati.
- Può essere utilizzata per dividere un unico grande gruppo di dati in più gruppi di dati durante il processo di esecuzione, riducendo significativamente la dimensione delle transazioni elaborate dal carico.
- L'opzione di caricamento a percorso diretto può servire per eseguire le operazioni di carico in modo molto rapido.

Per iniziare a utilizzare SQL\*Loader, per prima cosa è necessario acquisire una certa familiarità con il control file, come descritto nel prossimo paragrafo.

## 21.1 Il control file

Il control file comunica a Oracle in che modo leggere e caricare i dati. Il control file comunica a SQL\*Loader dove trovare i dati di origine da caricare e le tabelle in cui caricare questi dati, insieme a qualsiasi altra regola che è necessario applicare durante il processo di carico. Queste regole possono includere limitazioni alle eliminazioni (simili alle clausole `where` per le query) e istruzioni per unire più righe fisiche di un file di input in un'unica riga durante l'esecuzione di un comando `insert`. SQL\*Loader si servirà del control file per creare i comandi `insert` eseguiti per il caricamento dei dati.

Il control file viene creato a livello del sistema operativo, utilizzando qualsiasi editor di testo che consente di salvare i file come solo testo. Nel control file i comandi non devono obbedire a rigorosi requisiti di formattazione, tuttavia la standardizzazione della sintassi dei comandi semplificherà la futura manutenzione del control file.

Il listato seguente mostra un control file di esempio per il caricamento dei dati nella tabella `BIBLIOTECA`:

```
LOAD DATA
INFILE 'bookshelf.dat'
INTO TABLE BIBLIOTECA
(Titolo      POSITION(01:100)    CHAR,
 Editore     POSITION(101:120)   CHAR,
 NomeCategoria POSITION(121:140) CHAR,
 Valutazione   POSITION(141:142)  CHAR)
```

In questo esempio, i dati vengono caricati dal file `bookshelf.dat` nella tabella `BIOBLOTECA`. Il file `bookshelf.dat` contiene dati per tutte e quattro le colonne di `BIBLIOTECA`, con degli spazi vuoti che spaziano i caratteri non utilizzati in questi campi. Pertanto il valore della colonna `Editore` inizia sempre allo spazio 101 nel file, anche se il valore `Titolo` è inferiore ai 100 caratteri. Anche se questa formattazione ingrandisce il file di input, potrebbe semplificare il processo di caricamento. Per i campi non è necessario specificare la lunghezza, dato che le posizioni iniziale e finale nel flusso dei dati di input indicano effettivamente la lunghezza del campo.

La clausola `infile` assegna un nome al file di input, mentre la clausola `into table` specifica la tabella in cui verranno caricati i dati. Ogni colonna viene elencata, insieme alla posizione dove si trovano i propri dati in ciascun record fisico del file. Questo formato permette di caricare i dati anche se l'ordine della colonna dei dati di origine non corrisponde a quello delle colonne nella tabella.

Per eseguire questo caricamento, l'utente che se ne incarica dovrà possedere il privilegio `INSERT` per la tabella `BIBLIOTECA`.

### Caricamento di dati a lunghezza variabile

Se le colonne nel file di input hanno lunghezze variabili, si potrà ricorrere ai comandi di SQL\*Loader per comunicare a Oracle in che modo stabilire il punto in cui termina un valore. Nell'esempio seguente, una virgola separa i valori di input:

```

LOAD DATA
INFILE 'bookshelf.dat'
BADFILE '/user/load/bookshelf.bad'
TRUNCATE
INTO TABLE BIBLIOTECA
FIELDS TERMINATED BY ","
(Titolo, Editore, NomeCategoria, Valutazione)

```

La clausola `fields terminated by ","` comunica a SQL\*Loader che durante il caricamento il valore di ogni colonna sarà terminato da una virgola. Ciò significa, che il file di input non deve essere largo necessariamente 142 caratteri per ogni riga, come nel primo esempio di caricamento. Le lunghezze delle colonne non sono specificate nel control file, perché verranno determinate durante il processo di caricamento.

In questo esempio, il nome del file non valido è specificato dalla clausola `badfile`. Generalmente, il nome del file non valido viene indicato solo quando si intende reindirizzare il file in un'altra directory.

Questo esempio mostra anche l'uso della clausola `truncate` in un control file. Quando SQL\*Loader esegue questo control file, la tabella `BIBLIOTECA` verrà troncata ancor prima dell'avvio del processo di caricamento. Dato che i comandi `truncate` non possono essere annullati, occorre prestare particolare attenzione quando si utilizza questa opzione. Oltre a `truncate`, si possono usare le opzioni seguenti:

- `append` Aggiunge delle righe alla tabella.
- `insert` Aggiunge delle righe a una tabella vuota. Se la tabella non è vuota, l'operazione di caricamento verrà interrotta causando un errore.
- `replace` Svuota la tabella e poi aggiunge le righe nuove. L'utente deve possedere il privilegio `DELETE` per la tabella.

## 21.2 Avvio del caricamento

Per eseguire i comandi nel control file, è necessario eseguire SQL\*Loader con i parametri appropriati. SQL\*Loader viene avviato con il comando `SQLldr` su richiesta del sistema operativo.

**NOTA** *L'eseguibile SQL\*Loader potrebbe essere costituito dal nome SQLldr seguito dal numero di una versione. Per il nome esatto, si consulti la documentazione Oracle specifica della propria piattaforma. Per Oracle9i, il file eseguibile dovrebbe avere come nome SQLldr.*

Quando si esegue `SQLldr`, è necessario specificare il control file, il nome utente e la password e altre informazioni di carico importanti, come mostrato nella Tabella 21.1.

Ogni operazione di carico deve avere un control file, perché nessuno dei parametri di input specifica le informazioni importanti per il carico, il file di input e la tabella caricata.

Volendo, esiste la possibilità di separare gli argomenti da `SQLldr` con delle virgolette: è sufficiente inserirli con parole chiave (come `userid` o `log`), seguite dal valore del parametro. Le parole chiave sono sempre seguite da un segno di uguale (`=`) e dall'argomento appropriato.

Se si omette la parola chiave `userid`, verrà richiesto di inserirla. Se dopo il segno di uguale compare una barra (/), verrà utilizzato un account identificato esternamente. Inoltre, per connet-

**Tabella 21.1** Opzioni di SQL\*Loader.

PAROLA CHIAVE DI SQLldr	DESCRIZIONE
Userid	Nome utente e password per il caricamento, separate da una barra.
Control	Nome del control file.
Log	Nome del log file.
Bad	Nome del bad file.
Discard	Nome del file di scarto.
Discardmax	Massimo numero di righe da scartare prima di interrompere il caricamento. Il funzionamento predefinito prevede di consentire lo scarto di tutte le righe.
Skip	Numero di righe logiche del file di input da saltare prima di iniziare a caricare i dati. In genere utilizzata durante i ricaricamenti dallo stesso file di input in seguito a un caricamento parziale. Il valore predefinito è 0.
Load	Numero di righe logiche da caricare. Il valore predefinito è tutte.
Errors	Numero di errori consentiti. Il valore predefinito è 50.
Rows	Numero di righe da confermare in una volta. Utilizzare questo parametro per ridurre la dimensione della transazione durante il caricamento. Il valore predefinito per i caricamenti a percorso convenzionale è 64; per i caricamenti a percorso diretto è tutte le righe.
Bindsize	Dimensione dell'array di bind a percorso convenzionale, in byte. Il valore predefinito dipende dal sistema operativo.
Silent	Elimina i messaggi durante il caricamento.
Direct	Usare il caricamento a percorso diretto. Il valore predefinito è FALSE.
Parfile	Nome del file dei parametri che contiene le specifiche dei parametri di caricamenti aggiuntivi.
Parallel	Eseguire il caricamento parallelo. Il valore predefinito è FALSE.
File	File da cui allocare gli extent (per il caricamento parallelo).
Skip_Unusable_Indexes	Consente il caricamento in tabelle che hanno indici in stati non utilizzabili. Il valore predefinito è FALSE.
Skip_Index_Maintenance	Interrompe il mantenimento degli indici per i caricamenti a percorso diretto, lasciandoli in stati non utilizzabili. Il valore predefinito è FALSE.
Readsize	Dimensione del buffer di lettura; il valore predefinito è 1MB.
External_table	Usare le tabelle esterne per il caricamento; il valore predefinito è NOT_USED; altri valori validi sono GENERATE_ONLY ed EXECUTE.
Columnarrayrows	Numero di righe per l'array di colonne a percorso diretto: il valore predefinito è 5.000.
Streamsize	Dimensione in byte dello stream buffer a percorso diretto: il valore predefinito è 256.000.
Multithreading	Un flag per indicare se dev'essere utilizzato il multithreading durante un caricamento a percorso diretto.
Resumable	Un flag TRUE/FALSE per attivare o disattivare le operazioni che possono essere riprese per la sessione corrente; il valore predefinito è FALSE.
Resumable_name	Identificatore di testo per l'operazione che può essere ripresa.
Resumable_timeout	Tempo di attesa per un'operazione che può essere ripresa; il valore predefinito è 7200 secondi.

tersi a un database remoto e caricare in esso dei dati si può utilizzare una stringa di specifica del database Oracle Net. Per esempio, il comando potrebbe iniziare in questo modo:

---

```
sqlldr userid=usernm/mypass@dev
```

La parola chiave **direct**, che richiama l'opzione di carico a percorso diretto, verrà descritta nel paragrafo “Caricamento a percorso diretto” più avanti in questo capitolo.

Il parametro **SILENT** indica a SQL\*LOADER di eliminare alcuni dati informativi:

- **HEADER** elimina l'intestazione di SQL\*LOADER.
- **FEEDBACK** elimina il feedback a ogni punto di esecuzione.
- **ERRORS** elimina il log (in un file di log) di tutti quei record che hanno causato un errore di Oracle, anche se il conteggio viene sempre registrato.
- **DISCARDS** elimina il log (in un file di log) di tutti i record che sono stati eliminati, anche se il conteggio viene sempre registrato.
- **PARTITIONS** disabilita la scrittura delle statistiche di pre-partizione nel file di log.
- **ALL** elimina tutte i dati informativi precedenti.

Se si inseriscono più dati informativi, li si dovrà separare con una virgola e racchiudere la lista tra parentesi. Per esempio, è possibile eliminare le informazioni header ed errors con l'impostazione della parola chiave seguente:

```
silent=(HEADER,ERRORS)
```

**NOTA** *I comandi nel control file si sovrappongono a quelli nella linea di comando chiamante.*

Si provi a caricare un gruppo di dati di esempio nella tabella BIBLIOTECA, che ha quattro colonne (Titolo, Editore, NomeCategoria e Classificazione). I dati da caricare si trovano in un file di nome bookshelf.txt, costituito da due record:

```
Good Record,Some Publisher, SAGGIADULTI, 3
Another Title, Some Publisher, ILLUSADULTI, 4
```

**NOTA** *Ogni linea termina con un ritorno a capo. Anche se l'ultimo valore della prima linea non è lungo come la colonna in cui viene caricato, la riga si fermerà al ritorno a capo.*

I dati sono separati da virgolette, e non si ha l'intenzione di eliminare i dati precedentemente caricati in BIBLIOTECA, quindi il control file avrà l'aspetto seguente:

```
LOAD DATA
INFILE 'bookshelf.txt'
APPEND
INTO TABLE BIBLIOTECA
FIELDS TERMINATED BY ","
(Titolo, Editore, NomeCategoria, Valutazione)
```

Si salvi questo file con il nome bookshelf.ctl, nella stessa directory in cui si trova il file di dati di input. Successivamente, si esegua SQL\*LOADER comunicandogli di usare il control file:

```
sqlldr practice/practice control=bookshelf.ctl log=bookshelf.log
```

Completata l'operazione di caricamento, si dovrebbe avere un record caricato correttamente e uno no. Il record caricato correttamente si troverà nella tabella BIBLIOTECA:

```
select Titolo
  from BIBLIOTECA
 where Editore like '%Editore':
```

TITOLO

-----  
Good Record

Verrà creato un file di nome bookshelf.bad che conterrà un record:

Another Title,Some Publisher,ILLUSADULTI,4

Perché il record è stato rifiutato? Si verifichi il file di log, bookshelf.log, che in parte dirà:

```
Record 2: Rejected - Error on table BIBLIOTECA.
ORA-02291: integrity constraint (PRATICA.CATFK) violated -
parent key not found
```

Table BIBLIOTECA:

```
1 Row successfully loaded.
1 Row not loaded due to data errors.
```

La riga 2, quella di “Another Title”, è stata rifiutata perché il valore della colonna NomeCategoria non rispettava le limitazioni di chiave esterna: ILLUSADULTI non è elencato come categoria nella tabella CATEGORIA.

Dato che le righe rifiutate vengono isolate nel file non valido, questo file potrà servire da input per un caricamento successivo dopo aver corretto i dati.

## Record logici e fisici

Nella Tabella 21.1 molte parole chiave fanno riferimento a righe “logiche”. Una riga *logica* è una riga inserita nel database. In base alla struttura del file di input, si potrebbero unire più righe fisiche per formare un’unica riga logica.

Per esempio, il file di input potrebbe avere l’aspetto seguente:

Good Record,Some Publisher, SAGGIADULTI, 3

In questo caso, tra il record fisico e quello logico così creato si stabilirebbe una relazione uno a uno. Tuttavia, il file di dati potrebbe assumere questo aspetto:

```
Good Record,
Some Publisher,
SAGGIADULTI,
3
```

Per unire i dati, si dovrà ricorrere alle regole di continuazione. In questo caso, i valori della colonna vengono divisi uno per linea in modo che ogni record logico possa avere un certo numero di record fisici. Per unirli, si utilizzi la clausola concatenate nel control file. In questo caso, si specificherebbe concatenate 4 per creare un’unica riga logica a partire dalle quattro righe fisiche.

La logica per la creazione di un unico record logico a partire da più record fisici può essere molto più complessa di una semplice concatenazione. La clausola continueif può servire per specificare le condizioni che causano la continuazione dei record logici. I dati di input possono essere manipolati ulteriormente allo scopo di creare più record logici partendo da un unico re-

cord fisico (grazie all'uso di più clausole `into table`). Si analizzi la sintassi del control file nella voce “SQLLDR” del Capitolo 42 di questo libro, nonché le note indicate nel paragrafo successivo.

SQL\*Loader può essere utilizzato per generare più inserimenti da un'unica riga fisica (analogia alla capacità di inserire più tabelle descritta nel Capitolo 15). Per esempio, si supponga che i dati di input siano denormalizzati con i campi Città e Precipitazione, mentre i dati di input sono nel formato Città, Precipitazione1, Precipitazione2, Precipitazione3. Il control file assomiglierebbe a quanto segue (ovviamente in base alle posizioni fisiche di arresto e di avvio dei dati nel file):

```
into table PRECIPITAZIONE
when Citta != ''
(Città      POSITION(1:5)      CHAR,
 Precipitazione POSITION(6:10)  INTEGER EXTERNAL)    -- 1st row
--
into table PRECIPITAZIONE
when Citta != ''
(Città      POSITION(1:5)      CHAR,
 Precipitazione POSITION(11:16)  INTEGER EXTERNAL)   -- 2nd row
--
into table PRECIPITAZIONE
when Citta != ''
(Città      POSITION(1:5)      CHAR,
 Precipitazione POSITION(16:21)  INTEGER EXTERNAL)   -- 3rd row
```

Si osservi che su ogni riga fisica operano clausole `into table` separate. In questo esempio, esse generano righe separate nella tabella PRECIPITAZIONE; potrebbero essere utilizzate anche per inserire delle righe in più tabelle.

### 21.3 Note sulla sintassi del control file

La sintassi completa per i control file di SQL\*Loader è mostrata nella voce “SQLLDR” del Capitolo 42, quindi non verrà ripetuta in questa sede.

Nella clausola `load` è possibile specificare il carico come `recoverable` o `unrecoverable`. La clausola `unrecoverable` si applica solamente al caricamento a percorso diretto e verrà descritta nel paragrafo “Messa a punto delle operazioni di carico dei dati”, più avanti in questo capitolo.

Oltre a utilizzare la clausola `concatenate`, si può ricorrere alla clausola `continueif` per controllare in che modo i record fisici vengono assemblati in record logici. La clausola `this` fa riferimento al record fisico corrente, mentre la clausola `next` si riferisce al record fisico successivo. Per esempio, all'inizio di ogni record fisico si potrebbe creare un carattere di continuazione a due caratteri. Se questo record dovesse essere concatenato a quello precedente, si imposti questo valore a `***`. Successivamente, si potrebbe utilizzare la clausola `continueif next (1:2)=***` per creare un singolo record logico partendo da più record fisici. Il carattere di continuazione `***` non farà parte del record unito.

La sintassi per la clausola `into table` comprende anche una clausola `when`. La clausola `when`, mostrata nel listato seguente, funge da filtro applicato alle righe prima che queste vengano inserite nella tabella. Per esempio, si può specificare:

```
when Valutazione > 3
```

per caricare solo quei libri che hanno valutazioni maggiori di 3 nella tabella. Qualunque riga non superi la condizione `when` verrà scritta nel file di scarto, che pertanto raccoglie le righe che

potranno essere impiegate per operazioni successive di carico, ma che non passavano l'impostaione attuale delle condizioni when. Si possono utilizzare più condizioni when, collegate con clausole and.

Se si effettua il caricamento di record a lunghezza variabile le cui ultime colonne non hanno sempre un valore, occorre utilizzare la clausola trailing nullcols. Quando si usa questa clausola, SQL\*Loader genererà dei valori NULL per queste colonne.

Come mostrato in un esempio precedente di questo capitolo, è possibile utilizzare la clausola fields terminated by per caricare dati a lunghezza variabile. Invece di essere terminati da un carattere, i campi possono essere terminated by whitespace, enclosed by caratteri oppure optionally enclosed by altri caratteri.

Per esempio, la voce seguente carica i valori di NomeAutore e imposta i valori come maiuscoli durante l'operazione di inserimento. Se il valore è vuoto, verrà inserito NULL:

```
NomeAutore POSITION(10:34) CHAR TERMINATED BY WHITESPACE
NULLIF NomeAutore=BLANKS "UPPER(:NomeAutore)"
```

Quando si caricano i valori del tipo di dati DATE, è possibile specificare un formato di visualizzazione per la data. Per esempio, nel caso in cui si abbia una colonna chiamata DataRestituzione e i dati in ingresso presentino il formato Mon-DD-YYYY nelle prime 11 posizioni del record, si potrebbe specificare la parte DataRestituzione del carico nel modo seguente:

```
DataRestituzione POSITION (1:11) DATE "Mon-DD-YYYY"
```

Nella clausola into table, la parola chiave recnum può essere usata per assegnare un numero di record a ciascun record logico mentre viene letto dal file di dati, e questo valore verrà inserito nella colonna assegnata della tabella. La parola chiave constant permette di assegnare un valore costante a una colonna durante l'operazione di caricamento. Per le colonne di caratteri, si racchiude il valore costante tra apici. Se si usa la parola chiave sysdate, la colonna selezionata verrà popolata con la data e l'ora correnti del sistema.

```
DataPrestito SYSDATE
```

Se invece si usa l'opzione sequence, SQL\*Loader manterrà una sequenza di valori durante il processo di caricamento. Man mano che i record vengono elaborati, il valore della sequenza verrà aumentato dell'incremento specificato. Se nell'opzione di sequenza si utilizza la parola chiave max, i valori della sequenza useranno il valore attualmente massimo della colonna come punto iniziale della sequenza. Il listato seguente mostra l'uso dell'opzione sequence:

```
Sqnum_col SEQUENCE(MAX,1)
```

Per una sequenza, si possono specificare anche un valore iniziale e uno di incremento da utilizzare durante gli inserimenti. L'esempio seguente inserisce dei valori che partono da 100, con un incremento di 2. Se durante l'inserimento una riga viene rifiutata, il suo valore di sequenza verrà ignorato.

```
Sqnum_col SEQUENCE(100,2)
```

Se si memorizzano dei numeri nelle colonne di tipo VARCHAR2, non si utilizzi l'opzione sequence per queste colonne. Per esempio, se la tabella contiene già i valori da 1 a 10 in una colonna VARCHAR2, allora il valore più grande in questa colonna sarà 9, ossia la stringa di caratteri più grande. L'uso di questa stringa come base per un'opzione sequence farà in modo

che SQL\*Loader cerchi di inserire un record con 10 come valore creato più recentemente, e questo potrebbe causare dei conflitti con il record già esistente.

I control file di SQL\*Loader sono in grado di supportare regole di logica e business complesse. Per esempio, i dati di input di una colonna contenente valori monetari potrebbero avere un valore decimale implicito; 9990 verrebbe inserito come 99.90. In SQL\*Loader, si potrebbe inserire questo valore eseguendo i calcoli durante il caricamento dei dati:

```
money_amount      position (20:28) external decimal(9) ":tax_amount/100"
```

Per ulteriori esempi su SQL\*Loader e control file di esempio, fare riferimento alla sezione “SQL\*Loader Case Studies” della *Oracle9i Utilities Guide*.

## 21.4 Gestione delle operazioni di caricamento dei dati

Il caricamento di grandi volumi di dati è un’operazione batch. Le operazioni batch non dovrebbero essere eseguite contemporaneamente con le piccole transazioni prevalenti in molte applicazioni di database. Se molti utenti eseguono contemporaneamente delle piccole transazioni in una tabella, le operazioni batch dovrebbero essere programmate nella tabella in modo da avvenire quando nessun utente accede alla tabella.

Oracle mantiene la *coerenza di lettura* per le query degli utenti. Se si eseguono le operazioni di SQL\*Loader nella tabella proprio quando altri utenti stanno effettuando una query su quella tabella, Oracle manterrà internamente le voci di undo per consentire a questi utenti di visualizzare i loro dati così come erano quando hanno eseguito la prima query dei dati. Per ridurre al minimo la quantità di lavoro che Oracle dovrà svolgere per mantenere la coerenza di lettura (e per rendere minimo il calo di prestazioni conseguente causato proprio da questo sovraccarico), è consigliabile pianificare le operazioni di caricamento dei dati a lungo termine in modo che vengano eseguite quando nel database si verificano poche altre attività. Si faccia particolare attenzione a evitare i conflitti con altri utenti che cercano di accedere alla medesima tabella.

Occorre fare in modo che il processo di caricamento dei dati sia facile da mantenere e riutilizzare, stabilendo delle linee guida per la struttura e il formato dei file di dati di input. Maggiore il livello di standardizzazione per i formati dei dati di input, e più semplice sarà il riutilizzo dei control file per le operazioni di carico dei dati. Per operazioni di caricamento programmate ripetutamente nella stessa tabella, l’obiettivo dovrebbe essere il riutilizzo in ogni occasione dello stesso control file. Dopo ogni caricamento, si dovranno ricontrillare e spostare il file di log, il file non valido, il file di dati e quello di scarto per evitare che vengano accidentalmente sovrascritti.

Nel control file è necessario ricorrere ai commenti per indicare qualsiasi funzione di elaborazione particolare venga eseguita. Per creare un commento all’interno del control file, la linea dovrà essere preceduta da due trattini, come mostrato nell’esempio seguente:

```
-- Limitare il caricamento ai dipendenti LA:  
when Posizione='LA'
```

Se i commenti sono stati inseriti correttamente nel control file, le possibilità che quest’ultimo venga riutilizzato durante le future operazioni di caricamento aumenteranno. Inoltre, si semplificherà la gestione dello stesso processo di caricamento dei dati, come descritto nel prossimo paragrafo.

## Ripetizione delle operazioni di caricamento dei dati

Le operazioni di caricamento dei dati non seguono sempre i piani. Nel caricamento dei dati sono coinvolte molte variabili, e non tutte sono sempre controllabili. Per esempio, il proprietario dei dati di origine potrebbe modificare la formattazione dei suoi dati, invalidando così una parte del control file. Le regole business potrebbero cambiare, comportando altre modifiche forzate. Le strutture del database e la disponibilità di spazio potrebbero cambiare, influendo ancora di più sulla capacità di caricare i dati.

In una situazione ideale, il caricamento dei dati avverrebbe correttamente oppure fallirebbe completamente. Tuttavia, in molti casi, il caricamento dei dati potrebbe avvenire parzialmente, complicando ulteriormente il processo di recupero. Se alcuni record sono stati inseriti nella tabella, il tentativo di inserirli nuovamente dovrebbe comportare una violazione della chiave primaria. Se si crea il valore della chiave primaria durante un'operazione di inserimento (con l'opzione `sequence`), queste righe potrebbero anche non fallire al secondo tentativo, quindi verrebbero inserite due volte.

Per stabilire dove non è riuscito un caricamento, occorre utilizzare il file di log, il quale registrerà i punti di commit così come gli errori incontrati. Tutti i record rifiutati dovrebbero essere nel file non valido oppure nel file di scarto. È possibile ridurre al minimo il lavoro di recupero facendo fallire l'operazione di caricamento se si incontrano molti errori. Per indurre il caricamento a interrompersi prima di incontrare un gran numero di errori, si utilizzi la parola chiave `errors` del comando SQLLDR. È inoltre possibile usare la parola chiave `discardmax` per limitare il numero di record rifiutati ammessi prima che l'operazione di caricamento si interrompa.

Se si imposta `errors` a 0, il primo errore incontrato causerà la mancata riussita del caricamento. Cosa accadrebbe se questo caricamento fallisse dopo l'inserimento di ben 100 record? Ci sarebbero due possibilità: identificare ed eliminare i record inseriti, quindi applicare nuovamente l'intero caricamento, oppure saltare i record inseriti correttamente. La parola chiave `skip` di SQLLDR può servire proprio per saltare i primi 100 record durante il suo processo di caricamento. Il caricamento proseguirà con il record 101 (che si spera sia stato corretto prima del nuovo tentativo di caricamento). Se non si riesce a identificare le righe che sono state appena caricate nella tabella, durante il processo di riavvio si dovrà ricorrere all'opzione `skip`.

Le impostazioni corrette di `errors` e `discardmax` dipendono dal caricamento. Se si ha il controllo totale del processo di caricamento dei dati, e i dati sono stati "puliti" in modo appropriato prima di essere estratti da un file di carico, il livello di tolleranza per gli errori e le eliminazioni potrebbe essere molto basso. D'altro canto, se non si ha il controllo sulla sorgente del file di dati di input, `errors` e `discardmax` dovranno essere impostate a valori sufficientemente elevati da consentire il completamento del processo di carico. Terminato il caricamento, si dovrà ricontrillare il file di log, correggere i dati nel file non valido e caricare di nuovo i dati usando il file non valido originale come il nuovo file di input. Se le righe non sono state eliminate in modo corretto, si dovrà effettuare un altro caricamento usando il file di scarto originale come nuovo file di input.

Una volta modificato il valore di `NomeCategoria` errato, si potrà eseguire nuovamente l'esempio del caricamento della tabella BIBLIOTECA con il file bookshelf.dat. Durante il nuovo caricamento, quando si usa il file di dati di input originale si hanno due possibilità:

- Saltare la prima riga specificando `skip=1` nella linea di comando SQLLDR.
- Cercare di caricare entrambe le righe, in modo che la prima fallisca poiché era già stata caricata (causando così una violazione della chiave primaria).

In alternativa, si può utilizzare il file non valido come nuovo file di dati di input senza preoccuparsi degli errori e delle righe ignorate.

## 21.5 Messa a punto delle operazioni di caricamento dei dati

Oltre a eseguire i processi di caricamento dei dati negli orari non di punta, è possibile adottare altri accorgimenti per migliorare le prestazioni di caricamento. I passaggi seguenti influiranno tutti sull'intero ambiente di database, quindi dovranno essere organizzati con l'amministratore di database. La messa a punto di un'operazione di carico dei dati non dovrebbe avere un influsso negativo sul database oppure sui processi che lo supporta.

Per prima cosa, i caricamenti di dati batch dovrebbero essere programmati in modo tale da avvenire quando il database è in modalità NOARCHIVELOG. Quando si trova in questa modalità, il database non tiene un archivio dei suoi redo log file in linea prima di sovrascriverli. L'eliminazione del processo di archiviazione migliora le prestazioni delle transazioni. Dal momento che i dati vengono caricati da un file, questi potranno essere ricreati caricando nuovamente il file di dati piuttosto che recuperandoli da un redo log file archiviato.

Tuttavia, la disattivazione della modalità NOARCHIVELOG potrebbe creare dei grossi problemi. Non si potrà eseguire un recupero del database in un determinato momento, a meno che l'archiviazione non sia stata abilitata. Se nel database vengono effettuate transazioni non batch, probabilmente si dovrà eseguire il database sempre in modalità ARCHIVELOG, anche durante i caricamenti. Inoltre, il passaggio dalla modalità ARCHIVELOG a quella NOARCHIVELOG richiede la chiusura dell'istanza. Se si passa l'istanza alla modalità NOARCHIVELOG, si esegue l'operazione di caricamento dei dati e quindi si riporta l'istanza alla modalità ARCHIVELOG, si dovrebbe eseguire un backup del database (fare riferimento al Capitolo 40) subito dopo il suo riavvio.

Invece di eseguire l'intero database in modalità NOARCHIVELOG, è possibile disattivare l'archiviazione per il processo di carico dei dati tramite la parola chiave `unrecoverable` in SQL\*Loader. L'opzione `unrecoverable` disattiva la scrittura di voci di redo log per le transazioni eseguite nell'ambito di un caricamento di dati. Si dovrebbe ricorrere a questa opzione solo se si potranno ricreare le transazioni dai file di input durante un recupero. Se si segue questa strategia, si dovrà disporre di uno spazio adeguato per memorizzare i vecchi file di input nel caso in cui siano necessari per i recuperi successivi. L'opzione `unrecoverable` è disponibile solo per i caricamenti a percorso diretto, come descritto nel prossimo paragrafo.

Invece di controllare l'attività redo log a livello del processo di carico, si potrà controllare a livello di tabella o di partizione. Se si definisce un oggetto come `nologging`, allora gli inserimenti a livello di blocco eseguiti dal caricamento a percorso diretto di SQL\*Loader e il comando `insert /*+ APPEND */` non genereranno voci di redo log.

Se l'ambiente operativo ha più processori, è possibile sfruttare le CPU eseguendo le operazioni di caricamento in parallelo. L'opzione `parallel` di SQL\*Loader, come descritto nel prossimo paragrafo, utilizza più processi contemporanei di caricamento dei dati per ridurre il tempo complessivo necessario per caricare i dati.

Oltre a questi approcci, si dovrebbe collaborare con il proprio amministratore di database per essere certi che l'ambiente e le strutture del database siano stati messi a punto in modo adeguato per le operazioni di carico dei dati. I lavori di messa a punto dovrebbero includere quanto segue.

- Allocare anticipatamente dello spazio per la tabella, in modo da ridurre al minimo le estensioni dinamiche durante le operazioni di caricamento.
- Allocare risorse di memoria sufficienti nelle aree di memoria condivise, compresa l'area di log buffer.

- Semplificare il processo di scrittura dei dati creando più processi Database Writer (DBWR) per il database.
- Rimuovere qualunque trigger inutile durante i caricamenti dei dati. Se possibile, disattivare o rimuovere i trigger prima del caricamento, ed eseguire manualmente le operazioni di trigger sui dati caricati dopo averli caricati.
- Rimuovere o disattivare qualunque conflitto inutile nella tabella. Per disattivare e riattivare dinamicamente i conflitti, si può utilizzare SQL\*Loader.
- Rimuovere qualsiasi indice nelle tabelle. Se i dati sono stati “puliti” in modo appropriato prima del caricamento dei dati, allora le verifiche di univocità e le convalide delle chiavi esterne non saranno necessarie durante i caricamenti. L’eliminazione degli indici prima dei caricamenti dei dati migliora in modo significativo le prestazioni.

Se si lasciano degli indici nella tabella durante un’operazione di caricamento dei dati, Oracle dovrà gestire e bilanciare di nuovo l’indice ogni volta che viene inserito un nuovo record. Più grande è il caricamento di dati, maggiore sarà il lavoro che Oracle dovrà svolgere per gestire gli indici associati. Se possibile, si dovrebbe prendere in considerazione l’eventualità di eliminare gli indici prima del caricamento per poi ricrearli una volta completato questo processo. L’unica occasione in cui gli indici non penalizzano le prestazioni del caricamento di dati è durante il caricamento a percorso diretto, come descritto nel paragrafo seguente.

## **Caricamento a percorso diretto**

Quando si inseriscono dei record, SQL\*Loader genera un grande numero di istruzioni `insert`. Per scongiurare il sovraccarico che accompagna l’uso di un gran numero di inserimenti, si potrebbe usare l’opzione a percorso diretto in SQL\*Loader. Questa opzione crea blocchi di dati preformati e li inserisce nella tabella. Di conseguenza, le prestazioni dell’operazione di carico possono migliorare in modo molto significativo. Per usare l’opzione a percorso diretto, non si devono eseguire funzioni sui valori che vengono letti dal file di input.

Gli indici nella tabella caricata verranno collocati in uno stato DIRECT LOAD temporaneo (è possibile eseguire la query dello stato dell’indice da `USER_INDEXES`). Oracle sposterà i vecchi valori di indice in un indice temporaneo che crea e gestisce direttamente. Una volta terminato il processo di caricamento, i vecchi valori di indice verranno uniti a quelli nuovi per creare un indice nuovo, e Oracle eliminerà l’indice temporaneo che aveva creato. Quando l’indice sarà di nuovo valido, il suo stato cambierà in VALID. Per ridurre al minimo la quantità di spazio necessario all’indice temporaneo, i dati dovranno essere ordinati in anticipo secondo le colonne indicizzate. Il nome dell’indice per il quale i dati vengono ordinati in anticipo dovrebbe essere specificato tramite una clausola `sorted indexes` nel control file.

Per utilizzare l’opzione a percorso diretto, occorre specificare:

`DIRECT=TRUE`

come parola chiave nella linea di comando `SQLLDR`, oppure includere questa opzione nel control file.

Se si usa l’opzione a percorso diretto, per migliorare le prestazioni del caricamento dei dati si può utilizzare la parola chiave `unrecoverable`. Questa ordina a Oracle di non creare voci di redo log per il caricamento. Se più avanti si avrà la necessità di recuperare il database, si dovrà eseguire nuovamente il caricamento dei dati per recuperare i dati della tabella. Tutti i caricamenti con percorso convenzionale sono recuperabili e tutti i caricamenti a percorso diretto sono recuperabili per default.

I caricamenti a percorso diretto sono più rapidi di quelli convenzionali, e anche i caricamenti a percorso diretto irrecuperabili sono comunque più veloci. Dato che l'esecuzione di caricamenti irrecuperabili influisce sulle operazioni di recupero, si dovranno valutare le conseguenze di questo impatto rispetto ai vantaggi, in termini di prestazioni, che si otterranno. Se l'ambiente hardware possiede delle risorse aggiuntive disponibili durante il processo di caricamento, si potrà utilizzare l'opzione di caricamento a percorso diretto parallel per suddividere il lavoro di caricamento dei dati tra più processi. Le operazioni a percorso diretto parallele potrebbero completare il lavoro di caricamento più rapidamente di un singolo caricamento a percorso diretto.

Invece di usare l'opzione parallel, si potrebbe suddividere la tabella da caricare (si veda il Capitolo 18). Dato che SQL\*Loader consente di caricare una singola partizione, si potrebbero eseguire più operazioni SQL\*Loader contemporaneamente per popolare le parti separate di una tabella suddivisa. Questo metodo richiede un maggiore lavoro di amministrazione del database (per configurare e gestire le varie partizioni), tuttavia garantisce più flessibilità nell'esecuzione in parallelo e nella programmazione delle operazioni di carico.

A partire da Oracle9i è possibile sfruttare la funzionalità di caricamento multithread per i caricamenti a percorso diretto per convertire gli array di colonne in buffer di stream, e per eseguire il caricamento di buffer di stream in parallelo. Per attivare questa soluzione, occorre utilizzare il parametro `streamsize` e il flag `multithreading`.

Il caricamento a percorso diretto potrebbe influire sulla quantità di spazio necessaria per i dati della tabella. Dato che il caricamento a percorso diretto inserisce blocchi di dati, esso non segue i metodi usuali impiegati per allocare lo spazio in una tabella. I blocchi vengono inseriti alla fine della tabella, dopo l'*high watermark*, che è il blocco più elevato in cui i dati della tabella siano mai stati scritti. Se si inseriscono 100 blocchi di dati in una tabella e poi si cancellano tutte le righe, l'*high watermark* della tabella sarà ancora impostato a 100. Se in seguito si esegue un'operazione di caricamento dei dati SQL\*Loader convenzionale, le righe verranno inserite nei blocchi già allocati. Se invece si effettua un caricamento a percorso diretto, Oracle inserirà nuovi blocchi di dati dopo il blocco numero 100, comportando un probabile aumento dello spazio da allocare per la tabella. L'unico modo di ridurre l'*highwater mark* di una tabella consiste nel troncarla (il che comporta la cancellazione di tutte le righe e l'impossibilità di annullare questa operazione), oppure nell'eliminarla per poi ricrearla. Per identificare i problemi di spazio prima di iniziare un'operazione di caricamento si dovrebbe collaborare con il proprio amministratore di database.

## 21.6 Altri miglioramenti di Oracle9i

Oltre alle caratteristiche descritte in precedenza in questo capitolo, le soluzioni SQL\*Loader supportano Unicode e i tipi di dati espansi. A partire da Oracle9i, SQL\*Loader è in grado di caricare tipi di dati interi e decimali compatti/suddivisi in zone tra piattaforme che hanno ordini di byte differenti e che accettano dati decimali compatti o suddivisi in zone in base a EBCDIC e codificati in formato IBM. SQL\*Loader offre anche supporto per il caricamento di colonne XML, il caricamento di tipi di oggetti con sottotipi (vedere il Capitolo 30), e per Unicode (set di caratteri UTF16). SQL\*Loader fornisce anche supporto nativo per i nuovi tipi di dati di Oracle9i legati all'intervallo, all'ora e alla data (fare riferimento al Capitolo 9).

Se un'operazione SQL\*Loader non riesce, la si potrebbe riprendere dal punto in cui si è interrotta grazie alle opzioni `resumable`, `resumable_name` e `resumable_timeout`. Per esempio, se il segmento in cui stava scrivendo il job SQL\*Loader non si può estendere, è possibile disattivare l'operazione di caricamento, risolvere il problema di allocazione dello spazio e riprendere l'operazione. La capacità di eseguire queste azioni dipende dalla configurazione del database; è ne-

cessario cooperare con il proprio amministratore di database per assicurarsi che le funzionalità necessarie a riprendere un lavoro interrotto siano state abilitate e che la cronologia degli undo sia mantenuta in modo adeguato ai propri obiettivi.

A partire da Oracle9*i* è possibile accedere ai file esterni come se questi fossero delle tabelle contenute nel database. Questa caratteristica di “tabella esterna”, descritta nel Capitolo 25, consente di evitare il caricamento di grandi volumi di dati nel database. La sintassi per le definizioni delle tabelle esterne assomiglia moltissimo a quella del control file di SQL\*Loader. Anche se sono certi aspetti sono decisamente limitate (per esempio, nelle tabelle esterne non si possono eseguire operazioni DML), le tabelle esterne dovrebbero essere considerate come delle alternative alle operazioni di caricamento dei dati. Per i dettagli sull’implementazione, si consulti il Capitolo 25.

## Capitolo 22

# Accesso a dati remoti

- 22.1 **Database link**
- 22.2 **Uso di sinonimi per la trasparenza di dislocazione**
- 22.3 **Uso della pseudocolonna User nelle viste**
- 22.4 **Collegamenti dinamici:  
uso del comando copy di SQLPLUS**
- 22.5 **Connessione a un database remoto**

**A** mano a mano che le dimensioni e il numero dei database crescono, si presenta la necessità di condividere i dati in essi contenuti. La condivisione dei dati richiede un metodo per l'individuazione e l'accesso ai dati stessi. In Oracle, gli accessi a dati remoti come query e aggiornamenti sono abilitati grazie ai database link. Come descritto in questo capitolo, i database link consentono agli utenti di considerare un gruppo di database distribuiti come se fosse un unico database integrato. Questo capitolo contiene inoltre informazioni sulle connessioni dirette a database remoti, come quelle utilizzate in applicazioni client-server.

## 22.1 Database link

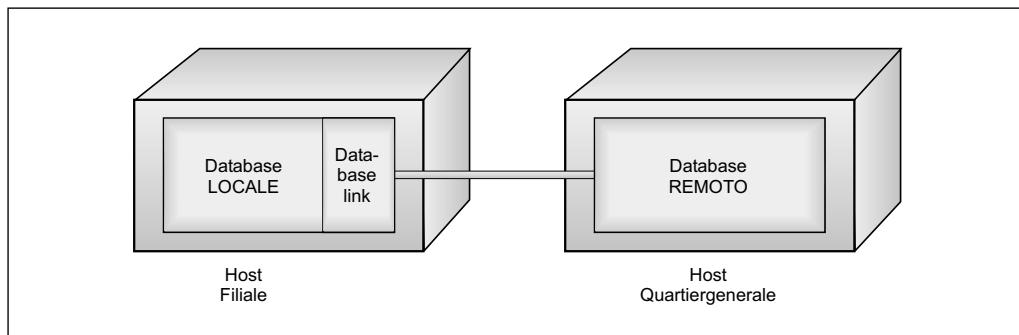
I *database link* comunicano a Oracle come passare da un database a un altro. Si può anche specificare il percorso di accesso con un metodo appropriato (consultare il paragrafo “Collegamenti dinamici: uso del comando copy di SQLPLUS”, più avanti nel capitolo). Se si prevede di utilizzare spesso la stessa connessione a un database remoto, un database link si rivelerà appropriato.

### Funzionamento di un database link

Un database link richiede che Oracle Net (noto in precedenza come SQL\*Net e Net8) venga eseguito su ciascuna delle macchine (*host*) coinvolte nell'accesso al database remoto. In genere Oracle Net viene avviato dall'amministratore del database (DBA) o dal gestore del sistema. Nella Figura 22.1 è riportato un esempio di architettura per un accesso remoto che utilizza un database link. Questa figura mostra due host, su entrambi dei quali viene eseguito Oracle Net. Su ciascuno degli host è presente un database. Un database link stabilisce una connessione dal primo database (denominato LOCALE, collocato sull'host Filiale) al secondo database (denominato REMOTO, collocato sull'host Quartiergenerale). Il database link mostrato nella Figura 22.1 si trova nel database “LOCALE”.

I database link specificano le informazioni di connessione seguenti:

- Il protocollo di comunicazione (per esempio TCP/IP) da utilizzare durante la connessione.
- L'host su cui risiede il database remoto.



**Figura 22.1** Esempio di architettura di un database link.

- Il nome del database sull'host remoto.
- Il nome di un account valido nel database remoto.
- La password per questo account.

Quando viene utilizzato, un database link si connette effettivamente al database remoto come utente e si disconnette non appena l'accesso ai dati remoti è completato. Un database link può essere *privato*, ovvero posseduto da un singolo utente, o *pubblico*, nel qual caso tutti gli utenti del database Locale possono utilizzare il link.

La sintassi per la creazione di un database link è descritta nel paragrafo “Sintassi per database link”, più avanti nel capitolo.

### Uso di un database link per query remote

Un utente del database Locale mostrato nella Figura 22.1 è in grado di accedere a oggetti nel database Remoto mediante un database link. A tal fine, è sufficiente aggiungere il nome del database link al nome di qualsiasi tabella o vista accessibile all'account remoto. Quando si aggiunge il nome del database link al nome di una tabella o di una vista, il nome del database link dovrà essere preceduto dal simbolo @.

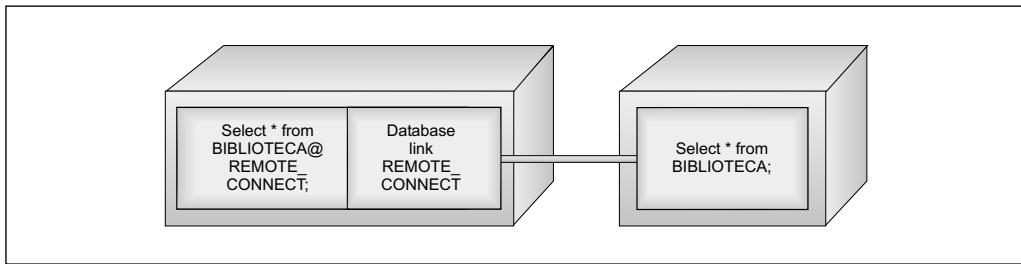
Per le tabelle locali, si dovrà fare riferimento al nome nella clausola from:

```
select * from BIBLIOTECA;
```

Per le tabelle remote, occorre utilizzare un database link di nome REMOTE\_CONNECT: Nella clausola from, si faccia riferimento al nome della tabella seguito da @REMOTE\_CONNECT:

```
select *
  from BIBLIOTECA@REMOTE_CONNECT;
```

Quando nella query precedente viene utilizzato il database link, Oracle si connette al database specificato dal link utilizzando il nome utente e la password forniti dal collegamento. Successivamente, eseguirà una query sulla tabella BIBLIOTECA in questo account e restituirà i dati all'utente che ha inizializzato la query. Queste operazioni sono rappresentate graficamente nella Figura 22.2. Il database link REMOTE\_CONNECT, mostrato in questa figura, si trova nel database Locale.



**Figura 22.2** Uso di un database link per una query remota.

Come illustrato nella Figura 22.2, la connessione al database Locale e l'uso del database link REMOTE\_CONNECT nella clausola from restituiscono gli stessi risultati di una connessione diretta al database remoto e dell'esecuzione della query senza il database link. Il database remoto appare come locale.

**NOTA** *Il numero massimo di database link che possono essere utilizzati in una singola query è impostato mediante il parametro OPEN-LINKS nel file dei parametri di inizializzazione del database. Tale parametro è normalmente impostato al valore 4.*

Esistono delle restrizioni alle query eseguite con i database link. Occorre evitare di usare i database link in query che utilizzano le parole chiave connect by, start with e prior. Alcune query che ricorrono a queste parole chiave possono funzionare (per esempio, se prior non viene utilizzata all'esterno della clausola connect by e start with non utilizza una subquery), ma la maggior parte delle query strutturate ad albero non funzionerà con i database link.

## Uso di un database link per sinonimi e viste

È possibile creare sinonimi e viste locali che fanno riferimento a oggetti remoti. A tal fine, è sufficiente creare un riferimento al nome del database link, preceduto dal simbolo “@”, ogni volta che si fa riferimento a una tabella remota. L'esempio seguente mostra come procedere per i sinonimi. In questo esempio, il comando create synonym viene eseguito da un account nel database Locale.

```
create synonym BIBLIOTECA_SYN
  for BIBLIOTECA_CONNECT;
```

In questo esempio, viene creato il sinonimo BIBLIOTECA\_SYN per la tabella BIBLIOTECA cui si accede mediante il database link REMOTE\_CONNECT. Ogni volta che questo sinonimo verrà utilizzato nella clausola from di una query, verrà eseguita una query sul database remoto. Si tratta di un meccanismo molto simile a quello delle query remote descritte in precedenza. L'unica vera differenza sta nel fatto che il database link ora è definito come parte di un oggetto locale (in questo caso di un sinonimo).

Cosa accade se l'account remoto cui si accede con il database link non possiede la tabella cui si fa riferimento? In questa situazione, si potrà utilizzare qualunque sinonimo disponibile per l'account remoto (sia pubblico che privato). Se questi sinonimi non esistono per una tabella per la quale è stato concesso l'accesso all'account remoto, è necessario specificare nella query il nome del proprietario della tabella, come mostrato nell'esempio seguente:

```
create synonym BIBLIOTECA_SYN
  for Pratica.BIBLIOTECA@REMOTE_CONNECT;
```

In questo esempio, l'account remoto utilizzato dal database link non possiede la tabella BIBLIOTECA, né ha un sinonimo con lo stesso nome. Tuttavia, l'account remoto dispone di alcuni privilegi sulla tabella BIBLIOTECA di proprietà dell'utente remoto Pratica nel database Remoto. Per questo motivo, vengono specificati il proprietario e il nome della tabella: entrambi verranno interpretati nel database Remoto. La sintassi per queste query e sinonimi è in gran parte la stessa che si utilizzerebbe se tutto si trovasse nel database locale: l'unica aggiunta è costituita dal nome del database link.

Per utilizzare un database link in una vista, è sufficiente aggiungerlo come suffisso ai nomi delle tabelle nel comando `create view`. L'esempio riportato di seguito crea una vista nel database locale di una tabella remota utilizzando il database link REMOTE\_CONNECT:

```
create view VISTA_BIBLIOTECA_LOCALE
as
select * from BIBLIOTECA@REMOTE_CONNECT
  where Titolo < 'M';
```

La clausola `from` in questo esempio fa riferimento a BIBLIOTECA@REMOTE\_CONNECT. Di conseguenza, la tabella base per questa vista non si trova nello stesso database della vista. Si noti osservi che nella query è stata inserita una clausola `where`, per limitare il numero di record restituiti dalla query per la vista.

Ora questa vista potrà essere trattata come qualsiasi altra vista del database locale. Si potrà concedere agli altri utenti l'accesso a questa vista, purché questi utenti possano accedere anche al database link REMOTE\_CONNECT.

## Uso di un database link per aggiornamenti remoti

La sintassi del database link per gli aggiornamenti remoti è la stessa utilizzata per le query remote. Occorre aggiungere il nome del database link al nome della tabella che si intende aggiornare. Per esempio, per modificare i valori di Classificazione per i libri in una tabella remota BIBLIOTECA, si esegue il comando `update`, riportato nel listato seguente:

```
update BIBLIOTECA@REMOTE_CONNECT
  set Valutazione = '5'
  where Titolo = 'INNUMERACY';
```

Questo comando `update` utilizzerà il database link REMOTE\_CONNECT per connettersi al database remoto. Quindi, aggiornerà la tabella BIBLIOTECA in questo database, basandosi sulle condizioni `set` e `where` specificate.

È possibile utilizzare delle subquery nella porzione `set` del comando `update` (si rimanda al Capitolo 15). La clausola `from` di tali subquery può fare riferimento sia al database locale sia al database remoto. Per fare riferimento al database remoto in una subquery, occorre aggiungere il nome del database link ai nomi delle tabelle nella clausola `from` della subquery. Di seguito è riportato un esempio:

```
update BIBLIOTECA@REMOTE_CONNECT      /*nel database remoto*/
  set Valutazione =
    (select valutazione
     from BIBLIOTECA@REMOTE_CONNECT /*nel database remoto*/
```

```

        where Titolo = 'WONDERFUL LIFE')
where Titolo = 'INNUMERACY';

```

**NOTA** Se non si aggiunge il nome del database link ai nomi delle tabelle nella clausola from delle subquery update, vengono utilizzate le tabelle nel database locale. Ciò vale anche se la tabella aggiornata si trova in un database remoto.

In questo esempio, la tabella remota BIBLIOTECA viene aggiornata basandosi sul valore Valutazione nella tabella remota BIBLIOTECA. Se nella subquery non viene utilizzato il database link, come nell'esempio seguente, al suo posto viene utilizzata la tabella BIBLIOTECA del database locale. Se questo non è intenzionale, i dati locali verranno mischiati nella tabella del database remoto. Se invece si intende realmente procedere in questo modo, è consigliata un'estrema cautela.

```

update BIBLIOTECA@REMOTE_CONNECT      /*nel database remoto*/
  set valutazione =
    (select Valutazione
     from BIBLIOTECA                  /*nel database locale*/
       where Titolo = 'WONDERFUL LIFE')
  where Titolo = 'INNUMERACY';

```

## Sintassi per i database link

Un database link può essere creato con il comando seguente:

```

create [shared] [public] database link REMOTE_CONNECT
connect to {current_user | nomeutente identified by password [authentication clause]}
using 'stringa connessione';

```

La sintassi specifica da utilizzare per la creazione di un database link dipende da due criteri:

- Lo stato “pubblico” o “privato” del database link.
- L’uso di connessioni predefinite o esplicite per il database remoto.

Questi criteri e la sintassi ad essi associata sono descritti nei paragrafi seguenti.

**NOTA** Per creare un database link, è necessario disporre del privilegio di sistema CREATE DATABASE LINK. L’account cui ci si connette nel database remoto dovrà disporre del privilegio di sistema CREATE SESSION. Entrambi i privilegi di sistema sono compresi nel ruolo CONNECT di Oracle.

## Database link pubblici e privati a confronto

Un database link *pubblico* è disponibile per tutti gli utenti di un database. Al contrario, un database link *privato* è disponibile solo per l’utente che lo ha creato. Un utente non può concedere ad altri l’accesso a un database link privato. Il database link può essere pubblico (disponibile per tutti gli utenti) o in alternativa privato.

Per definire un database link come pubblico, si utilizzi la parola chiave public nel comando create database link, come mostrato nell’esempio seguente.

```

create public database link REMOTE_CONNECT
connect to nomeutente identified by password
  using 'stringa connessione';

```

**NOTA** Per creare un database link pubblico, è necessario disporre del privilegio di sistema CREATE PUBLIC DATABASE LINK. Questo privilegio è compreso nel ruolo DBA di Oracle.

### Connessioni predefinite a confronto con connessioni esplicite

Al posto della clausola connect to ... identified by ..., si può utilizzare connect to current\_user quando si crea un database link. Se si usa l'opzione current\_user, quando utilizzato, questo collegamento cercherà di aprire una sessione nel database remoto che ha lo stesso nome utente e la stessa password dell'account nel database locale. In questo caso, si parla di *connessione predefinita*, poiché la combinazione nome utente/password sarà impostata in modo predefinito alla combinazione in uso nel database locale.

Il listato seguente mostra l'esempio di un database link pubblico creato con una connessione predefinita (l'uso delle connessioni predefinite viene spiegato più in dettaglio nel paragrafo "Utilizzo della pseudocolonna Utente nelle viste" più avanti nel capitolo):

```
create public database link REMOTE_CONNECT
connect to current_user
using 'stringa connessione';
```

Se utilizzato, questo database link cercherà di connettersi al database remoto utilizzando il nome utente e la password dell'utente corrente. Se nel database remoto il nome utente corrente non è valido, o se la password è diversa, il tentativo di connessione non riuscirà. Ciò comporterà anche il fallimento dell'istruzione SQL che utilizza il collegamento.

Una connessione *esplicita* specifica un nome utente e una password che il database link utilizzerà durante la connessione al database remoto. A prescindere da quale account locale utilizzi il collegamento, verrà usato sempre lo stesso account remoto. Di seguito è riportata la creazione di un database link con connessione esplicita:

```
create public database link REMOTE_CONNECT
connect to WAREHOUSE identified by ACCESS339
using 'stringa connessione';
```

Questo esempio mostra un uso comune delle connessioni esplicite nei database link. Nel database remoto è stato creato un utente di nome Warehouse, cui è stata assegnata la password ACCESS339. All'account Warehouse si potrà quindi concedere l'accesso SELECT a tabelle specifiche, esclusivamente per l'utilizzo mediante database link. Il database link REMOTE\_CONNECT fornirà quindi a tutti gli utenti locali l'accesso all'account Warehouse.

### Sintassi della stringa di connessione

Oracle Net usa *nomi di servizio* per identificare le connessioni remote. I dettagli di connessione per questi nomi di servizio sono contenuti in file distribuiti a ciascun host della rete. Quando incontra un nome di servizio, Oracle controlla il file di configurazione locale di Oracle Net (chiamato tnsnames.ora) per determinare quale protocollo, nome host e nome database utilizzare durante la connessione. Tutte le informazioni di connessione si trovano in file esterni.

Quando si usa Oracle Net, è necessario conoscere il nome del servizio che punta al database remoto. Per esempio, se il nome di servizio 'HQ' specifica i parametri di connessione per il database necessario, allora 'HQ' dovrà essere utilizzato come stringa di connessione nel comando create database link. L'esempio seguente mostra un database link privato che utilizza una connessione predefinita e un nome di servizio di Oracle Net:

---

```
create database link REMOTE_CONNECT
connect to current_user
using 'HQ';
```

Quando si utilizza questo collegamento, Oracle verifica il file tnsnames.ora sull'host locale per determinare il database cui connettersi. Quando cerca di connettersi a questo database, Oracle usa il nome utente e la password dell'utente corrente.

I file tnsnames.ora per una rete di database dovrebbero essere coordinati dai DBA. Nel listato seguente, è riportata una tipica voce del file tnsnames.ora (per una rete che utilizza il protocollo TCP/IP).

```
HQ =(DESCRIPTION=
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL=TCP)
          (HOST=host1)
          (PORT=1521))
      )
      (CONNECT_DATA=
        (SERVICE_NAME = HQ.host1)
      )
    )
```

In questo listato, il nome di servizio HQ viene associato a un descrittore di connessione che indica al database quale protocollo utilizzare (TCP/IP), e a quale host (host1) e database (HQ) connettersi. L'informazione di "porta" fa riferimento alla porta sull'host che verrà utilizzata per la connessione; questo dato è specifico dell'ambiente. Protocolli differenti avranno parole chiave differenti, ma tutti dovranno trasmettere lo stesso contenuto.

### **Uso di database link condivisi**

Se per le connessioni di database si usa l'opzione Shared Server e le applicazioni usano più connessioni di database link concorrenti, l'impiego di *database link condivisi* potrebbe rappresentare un vantaggio. Un database link condiviso ricorre a connessioni server condivise per supportare le connessioni dei database link. Se più database link accedono contemporaneamente a un database remoto, si potranno utilizzare i database link condivisi per ridurre il numero di connessioni server necessarie.

Per creare un database link condiviso, occorre usare la parola chiave **shared** del comando **create database link**. Come mostra il listato seguente, si dovrà specificare anche uno schema e una password per il database remoto:

```
create shared database link HR_LINK_SHARED
connect to current_user
authenticated by HR identified by puffin55556d
using 'hq';
```

Il database link HR\_LINK\_SHARED usa il nome utente e la password dell'utente connesso quando accede al database "hq", come specificato tramite la clausola **connect to current\_user**. Per impedire a utenti non autorizzati di utilizzare il collegamento condiviso, questo richiede la clausola **authenticated by**. In questo esempio, per l'autenticazione viene impiegato un account di applicazione, ma si potrebbe ricorrere anche a uno schema vuoto. L'account di autenticazione deve disporre del privilegio di sistema **CREATE SESSION**. Mentre si usa il database link HR\_LINK\_SHARED, i tentativi di connessione includeranno l'autenticazione per l'account del collegamento HR.

Se si modifica la password sull'account di autenticazione, si dovrà rimuovere e creare di nuovo ogni database link che faccia riferimento a questo account. Per semplificare la gestione, si crei un account che viene impiegato solo per l'autenticazione di connessioni di database link condivisi. L'account dovrebbe avere solamente il privilegio di sistema CREATE SESSION, inoltre non dovrebbe disporre di privilegi su nessuna delle tabelle dell'applicazione.

Se l'applicazione utilizza raramente i database link, sarebbe opportuno usare i database link tradizionali senza la clausola shared. Infatti, senza questa clausola ogni connessione di database link richiede una connessione separata al database remoto.

## 22.2 Uso di sinonimi per la trasparenza di dislocazione

Durante la vita di un'applicazione, è molto probabile che i suoi dati vengano spostati da un database a un altro o da un host a un altro. Di conseguenza, la gestione di un'applicazione risulta semplificata se l'esatta dislocazione fisica di un oggetto del database viene nascosta all'utente (e all'applicazione).

Il modo migliore per integrare tale trasparenza di dislocazione è rappresentato dall'uso di sinonimi. Invece di scrivere applicazioni (o report SQLPLUS) che contengono query in cui sia specificato il proprietario di una tabella, come:

```
select *
  from Pratica.BIBLIOTECA;
```

si potrebbe creare un sinonimo per detta tabella e poi nella query fare riferimento a questo sinonimo:

```
create synonym BIBLIOTECA
  for Pratica.BIBLIOTECA;
```

```
select *
  from BIBLIOTECA;
```

La logica richiesta per l'individuazione dei dati viene così spostata all'esterno dell'applicazione e all'interno del database. Lo spostamento della logica di dislocazione della tabella verso il database rappresenterà un vantaggio ogni volta che la tabella stessa verrà spostata (per esempio, per il passaggio da un database di sviluppo a un database di verifica).

Oltre a nascondere la proprietà delle tabelle a un'applicazione, è anche possibile nascondere la dislocazione fisica dei dati grazie a database link e sinonimi. Utilizzando sinonimi locali per tabelle remote, un altro livello di logica viene spostato all'esterno dell'applicazione e all'interno del database. Per esempio, il sinonimo locale BIBLIOTECA, come definito nel listato seguente, fa riferimento a una tabella collocata in un database differente, su un host diverso. Se questa tabella verrà spostata, sarà sufficiente modificare solo il collegamento, mentre il codice applicativo che utilizza il sinonimo resterà invariato.

```
create synonym BIBLIOTECA
  for BIBLIOTECA@REMOTE_CONNECT;
```

Se l'account remoto utilizzato dal database link non è il proprietario dell'oggetto cui si fa riferimento, vi sono due possibilità. Per prima cosa è possibile fare riferimento a un sinonimo disponibile nel database remoto:

---

```
create synonym BIBLIOTECA
  for BIBLIOTECA@REMOTE_CONNECT;
```

in cui BIBLIOTECA, nell'account remoto utilizzato dal database link, è un sinonimo per la tabella BIBLIOTECA di un altro utente.

La seconda possibilità consiste nell'includere il nome del proprietario remoto durante la creazione del sinonimo locale, come nel listato seguente.

```
create synonym BIBLIOTECA
  for Pratica.BIBLIOTECA@REMOTE_CONNECT;
```

Questi due esempi avranno la stessa funzionalità per le query, ma con alcune differenze. Il secondo, che comprende il nome del proprietario, comporta una gestione potenzialmente più difficile, in quanto non viene utilizzato un sinonimo nel database remoto. I due esempi presentano anche leggere differenze di funzionalità quando viene utilizzato il comando describe. Se l'account remoto accede a un sinonimo (invece che a una tabella) non sarà possibile descrivere questa tabella, anche se resterà possibile effettuare selezioni. Perché il comando describe funzioni correttamente, è necessario utilizzare il formato mostrato nell'ultimo esempio e specificare il proprietario.

## 22.3 Uso della pseudocolonna User nelle viste

La pseudocolonna User è molto utile quando si impiegano metodi di accesso a dati remoti. Per esempio, si potrebbe desiderare di fare in modo che non tutti gli utenti remoti siano in grado di visualizzare tutti i record di una tabella. Per risolvere questo problema, è necessario pensare agli utenti remoti come utenti speciali all'interno del database. Per imporre delle restrizioni sui dati, è necessario creare una vista cui accederanno gli account remoti. Ma che cosa si può utilizzare nella clausola where per applicare adeguate restrizioni ai record? La pseudocolonna User, in combinazione con nomi utente adeguatamente selezionati, consentirà di imporre tali restrizioni.

Come descritto nel Capitolo 19, le query utilizzate per definire le viste possono anche fare riferimento a *pseudocolonne*. Una pseudocolonna è una “colonna” che restituisce un valore quando viene selezionata, ma non è una vera colonna di una tabella. Quando selezionata, la pseudocolonna User restituisce sempre il nome utente Oracle che ha eseguito la query. Quindi, se una colonna della tabella contiene nomi utente, questi valori potranno essere confrontati con la pseudocolonna User per applicare restrizioni ai relativi record, come mostrato nell'esempio seguente. In questo esempio, viene eseguita una query sulla tabella NOME. Se il valore della prima parte della colonna Nome coincide con il nome dell'utente che inserisce la query, vengono restituiti i record.

```
create view MIO_PRESTITO as
select * from BIBLIOTECA.PRESTITO
where SUBSTR(Nome,1,INSTR(Nome,' ')-1) = User;
```

**NOTA** Per questa discussione è necessario cambiare punto di vista. Dato che al momento si sta parlando di operazioni sul database che possiede la tabella su cui si effettua la query, questo database viene indicato come “locale”, mentre gli utenti di altri database sono considerati come “remoti”.

Quando si applicano restrizioni all'accesso remoto a righe della tabella, in primo luogo è necessario considerare quali colonne è opportuno utilizzare per tali restrizioni. Solitamente, esi-

stono divisioni logiche nei dati di una tabella, come Reparto o Area. Per ciascuna singola divisione, occorre creare un account utente separato nel database locale. Per questo esempio, si aggiunga una colonna Area alla tabella BIBLIOTECA. Ora, si potrà registrare l'elenco dei libri da più dislocazioni distribuite in un'unica tabella:

```
alter table BIBLIOTECA
add
(Area VARCHAR2(10));
```

Si supponga che nella tabella BIBLIOTECA siano rappresentate quattro aree principali e che per ciascuna di esse sia stato creato un account Oracle. Sarà quindi possibile configurare ciascun database link dell'utente remoto in modo che utilizzi il proprio account utente specifico nel database locale. Per questo esempio, si supponga che le aree si chiamino NORD, EST, SUD e OVEST. Per ciascuna area si dovrà creare un database link specifico. Per esempio, i membri del reparto SUD utilizzerebbero il database link riportato nel listato seguente:

```
create database link SUD_LINK
connect to SUD identified by PAW
using 'HQ';
```

Il database link di questo esempio è privato con una connessione esplicita all'account SUD nel database remoto.

Quando gli utenti remoti eseguono query mediante i propri database link (come SUD\_LINK nell'esempio precedente), verranno connessi al database HQ, con il nome del proprio Reparto (come SUD) come nome utente. Così, il valore della colonna Utente, per qualsiasi tabella su cui l'utente esegue una query, sarà 'SUD'.

Ora si crei una vista della tabella base, mettendo a confronto la pseudocolonna User con il valore della colonna Area nella clausola where della vista (questo particolare impiego della pseudocolonna Utente era già stato dimostrato nel Capitolo 19):

```
create or replace view BIBLIOTECA_LIMITATA
as select *
from BIBLIOTECA
where Area = User;
```

Un utente che si connette mediante il database link SUD\_LINK, e viene così connesso come utente SUD, visualizzerebbe solo i record di BIBLIOTECA che hanno un valore di Area uguale a 'SUD'. Se gli utenti accedono alla tabella da un database remoto, le loro connessioni avvengono attraverso database link e gli account locali da essi utilizzati sono noti all'operatore poiché li ha configurati.

Questo tipo di restrizione può essere imposto anche nel database remoto, invece che nel database in cui risiede la tabella. Gli utenti nel database remoto possono creare viste nei propri database con il formato seguente:

```
create or replace view SUD_BIBLIOTECA
as select *
from BIBLIOTECA@REMOTE_CONNECT
where Area = 'SUD';
```

In questo caso, le restrizioni di Area restano ancora in vigore, tuttavia sono amministrate localmente e codificate nella query della vista. La scelta tra le due possibilità di restrizione (locale o remota) dipende dal numero di account necessari per imporre la restrizione desiderata.

Per proteggere il database di produzione, si dovrebbero limitare i privilegi concessi agli account utilizzati dai database link. Inoltre, sarebbe opportuno concedere questi privilegi tramite i ruoli, e utilizzare le viste (con la clausola *with read only* o *with check option*) per limitare ulteriormente la possibilità che questi account vengano utilizzati per apportare modifiche non autorizzate ai dati.

## 22.4 Collegamenti dinamici: uso del comando copy di SQLPLUS

Il comando *copy* di SQLPLUS è poco utilizzato e sottovalutato. Esso consente la copia di dati tra database (o all'interno dello stesso database) mediante SQLPLUS. Anche se permette di selezionare le colonne da copiare, questo comando funziona al meglio quando vengono selezionate tutte le colonne di una tabella. In breve, il vantaggio più grande nell'utilizzo di questo comando consiste nella sua capacità di applicare il comando *commit* dopo che ciascun array di dati è stato elaborato. Ciò genera a sua volta transazioni di dimensioni maneggevoli.

Si consideri il caso di una tabella di grandi dimensioni come *BIBLIOTECA\_PRESTITO*. Come si procede se la tabella *BIBLIOTECA\_PRESTITO* contiene 100.000 righe che occupano uno spazio complessivo pari a 100 MB ed è necessario creare una copia della tabella in un database differente? Il metodo più semplice richiede la creazione di un database link e l'uso del medesimo in un comando *create table... as select*, come mostrato di seguito.

```
create database link REMOTE_CONNECT
connect to PRATICA identified by PRATICA
using 'HQ';

create table BIBLIOTECA_PRESTITO
as
select * from BIBLIOTECA_PRESTITO@REMOTE_CONNECT;
```

Il primo comando crea il database link, mentre il secondo crea una nuova tabella basata su tutti i dati della tabella remota.

Sfortunatamente, questo metodo porta alla creazione di una transazione molto grande (tutte le 100.000 righe verrebbero inserite nella nuova tabella come una singola transazione) che impone un carico pesante alle strutture interne di Oracle, chiamate *segmenti di rollback*. I segmenti di rollback e gli undo gestiti a livello di sistema (si consulti il Capitolo 40) memorizzano l'immagine dei dati fino a quando questi non vengono trasmessi al database. Dato che questa tabella viene popolata con un singolo inserimento, verrà generata un'unica transazione di grandi dimensioni, che potrebbe superare lo spazio nei segmenti di rollback attualmente disponibili. Tale insuccesso causerà a sua volta il fallimento della creazione della tabella.

Per suddividere la transazione in porzioni più piccole, occorre utilizzare il comando *copy* di SQLPLUS. La sintassi di questo comando è:

```
copy from
[nomeutente remoto/remoto password@stringa connessione]
[to nomeutente/password@stringa connessione]
{aggiungi|create|insert|replace}
nome tabella
using subquery;
```

Se la destinazione dei dati copiati è l'account corrente, la parola to, il nome utente locale, la password e la stringa di connessione non sono necessari. Se l'account corrente è l'origine dei dati copiati, non sono necessarie le informazioni di connessione remota per l'origine dei dati.

Per impostare le dimensioni delle voci di transazione si utilizza il comando set di SQLPLUS che serve per impostare un valore per il parametro arrayszie. Ciò consente di definire il numero di record che verranno recuperati in ciascun "batch". Il parametro copycommit specifica a SQLPLUS quanti batch dovranno essere trasmessi per volta. Lo script SQLPLUS riportato di seguito esegue la stessa copia dei dati consentita dal comando create table as; tuttavia, suddivide la singola transazione in più transazioni. Nell'esempio seguente, i dati sono trasmessi dopo ciascun gruppo di 1.000 record. In questo modo, si riducono le dimensioni necessarie per i segmenti di rollback della transazione da 100 MB a 1 MB.

```
set copycommit 1
set arrayszie 1000

copy from PRATICA/PRATICA@HQ -
create BIBLIOTECA_PRESTITO -
using -
select * from BIBLIOTECA_PRESTITO
```

**NOTA** *Ad eccezione dell'ultima, ciascuna linea nel comando copy deve terminare con un trattino (-), in quanto si tratta di un comando SQLPLUS.*

Le differenti opzioni relative ai dati all'interno del comando copy sono descritte nella Tabella 22.1.

Inizialmente, le informazioni restituite da questo comando possono confondere. Completato l'ultimo commit, il database indica all'utente il numero di record trasmessi durante l'ultimo batch. Il database non riporta il numero totale di record trasmessi (a meno che questi non siano stati trasmessi tutti in un singolo batch).

## 22.5 Connessione a un database remoto

Oltre alle connessioni tra database descritte in precedenza in questo capitolo, è anche possibile connettersi direttamente a un database remoto mediante uno strumento di Oracle. Così, invece di immettere:

**Tabella 22.1** Opzioni del comando copy.

OPZIONE	DESCRIZIONE
APPEND	Inserisce le righe nella tabella di destinazione. Crea automaticamente la tabella, se questa non esiste.
CREATE	Crea la tabella, quindi inserisce le righe.
INSERT	Inserisce le righe nella tabella di destinazione, se questa esiste. Altrimenti restituisce un errore. Per utilizzare INSERT, tutte le colonne devono essere specificate nella sottoquery "using".
REPLACE	Elimina la tabella di destinazione esistente e la sostituisce con una nuova tabella contenente i dati copiati.

```
sqlplus nomeutente/password
```

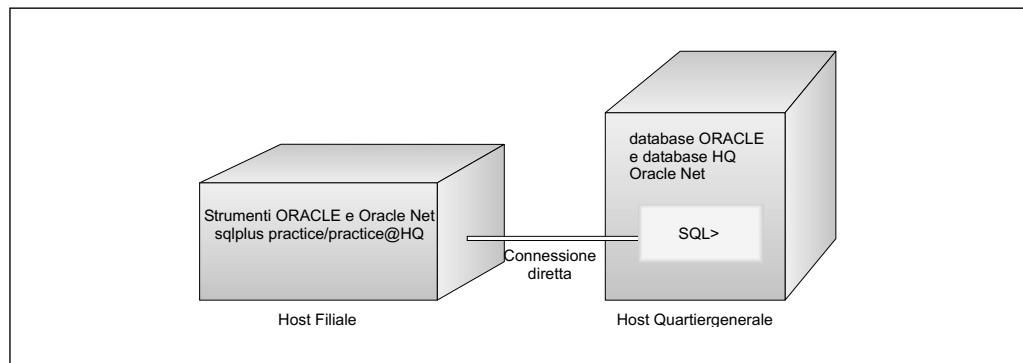
e accedere al database locale, è possibile accedere direttamente a un database remoto. A tal fine, occorre inserire il proprio nome utente e la propria password insieme alla stringa di connessione Oracle Net per il database remoto:

```
sqlplus nomeutente/password@HQ
```

Questo comando stabilisce una connessione diretta al database HQ. La configurazione dell'host per questo tipo di connessione è rappresentata nella Figura 22.3; l'host Filiale possiede gli strumenti di Oracle (come SQLPLUS) ed esegue Oracle Net, mentre l'host Remoto esegue Oracle Net e possiede un database Oracle. Sull'host Filiale può esserci come può non esserci un database; la specificazione della stringa di connessione Oracle Net al database remoto induce Oracle a ignorare qualsiasi database locale.

Come mostra la Figura 22.3, per l'host Filiale i requisiti hardware sono davvero minimi. Questo host deve supportare solo lo strumento front-end e Oracle Net, una configurazione tipica per applicazioni client-server. Una macchina client, come l'host Filiale, viene utilizzata principalmente per la presentazione dei dati mediante gli strumenti di accesso al database. Il lato del server, come l'host Quartier generali, viene utilizzato per la gestione dei dati e l'elaborazione delle richieste di accesso ai medesimi da parte degli utenti.

A prescindere dalla configurazione scelta e dagli strumenti di configurazione disponibili, è necessario comunicare a Oracle Net come trovare il database remoto. È opportuno collaborare con il proprio DBA per essere certi che il server remoto sia configurato in modo adeguato per ascoltare le nuove richieste di connessione, e per assicurarsi che le macchine client siano configurate correttamente in modo da inviare queste richieste.



**Figura 22.3** Esempio di architettura di una connessione remota.



## Capitolo 23

# Uso delle viste materializzate

- 23.1 **Funzionalità**
- 23.2 **Privilegi di sistema richiesti**
- 23.3 **Privilegi di tabella richiesti**
- 23.4 **Viste materializzate di sola lettura e viste materializzate aggiornabili a confronto**
- 23.5 **Sintassi di create materialized view**
- 23.6 **Aggiornamento delle viste materializzate**
- 23.7 **Sintassi di create materialized view log**
- 23.8 **Modifica dei log e delle viste materializzate**
- 23.9 **Eliminazione di viste materializzate e log**

Per migliorare le prestazioni di un'applicazione è possibile creare delle copie locali di tabelle remote che utilizzano dati distribuiti, oppure creare summary table basate su operazioni group by. Oracle mette a disposizione delle *viste materializzate* per memorizzare le copie di dati o di aggregazioni. Nelle versioni precedenti di Oracle, le viste materializzate basate su dati remoti erano note con il nome di “snapshot.” Le viste materializzate possono essere utilizzate per replicare un'intera tabella o parte di essa, oppure per replicare il risultato di una query eseguita su più tabelle; gli aggiornamenti dei dati replicati possono essere effettuati automaticamente dal database a intervalli di tempo definibili. Questo capitolo spiegherà l'uso generale delle viste materializzate, comprese le loro strategie di aggiornamento, seguite da una descrizione delle strategie di ottimizzazione disponibili.

## 23.1 Funzionalità

Le viste materializzate sono copie ( dette anche *repliche*) di dati, basate su query. Nella sua forma più semplice, una vista materializzata può essere considerata come una tabella creata da un comando come il seguente:

```
create table BIBLIOTECA_LOCALE  
as  
select * from BIBLIOTECAA@REMOTE_CONNECT;
```

In questo esempio, viene creata nel database locale una tabella di nome BIBLIOTECA\_LOCALE, che viene popolata con dati provenienti da un database remoto (definito dal database link REMOTE\_CONNECT). Dopo che la tabella BIBLIOTECA\_LOCALE è stata creata, i suoi dati potrebbero perdere immediatamente la sincronizzazione con la tabella principale (BIBLIOTECAA@REMOTE\_CONNECT). Inoltre, gli utenti locali possono aggiornare BIBLIOTECA\_LOCALE, complicando ulteriormente la sua sincronizzazione con la tabella principale.

Nonostante questi problemi di sincronia, la replica dei dati eseguita in questo modo porta notevoli vantaggi. La creazione di copie locali di dati remoti può migliorare le prestazioni delle query distribuite, in particolare se i dati delle tabelle non cambiano molto di frequente. Si potrebbe utilizzare il processo di creazione di tabelle locali anche per limitare il numero di righe o di colonne restituite, oppure per creare nuove colonne (per esempio, applicando le funzioni a valori selezionati). Si tratta di una strategia molto comune per ambienti di supporto decisionale, in cui query complesse servono per effettuare il “riepilogo” periodico dei dati in summary table da utilizzare durante le analisi.

Le viste materializzate automatizzano i processi di aggiornamento e di replica dei dati. Quando si creano le viste materializzate, viene fissato un *intervallo di aggiornamento* per programmare gli aggiornamenti dei dati replicati. Gli aggiornamenti locali possono essere evitati ripiegando su aggiornamenti basati sulle transazioni. Gli aggiornamenti basati sulle transazioni, disponibili per alcuni tipi di viste materializzate, inviano dal database master solo le righe che sono cambiate per la vista materializzata. Questa capacità, descritta più avanti nel capitolo, può migliorare in modo significativo le prestazioni degli aggiornamenti.

## 23.2 Privilegi di sistema richiesti

Per creare una vista materializzata, è necessario possedere i privilegi necessari a creare gli oggetti base che tale vista utilizzerà: si deve disporre dei privilegi CREATE MATERIALIZED VIEW o CREATE SNAPSHOT, così come dei privilegi di sistema CREATE TABLE o CREATE ANY TABLE. Inoltre, è necessario disporre del privilegio di sistema UNLIMITED TABLESPACE oppure di una specifica quota di spazio sufficiente in una tablespace locale. Per creare su richiesta una vista materializzata per aggiornamenti, occorre inoltre avere il privilegio di sistema ON COMMIT REFRESH su qualsiasi tabella di cui non si è proprietari, oppure il privilegio di sistema ON COMMIT REFRESH.

**NOTA** Oracle9i supporta la parola chiave snapshot al posto di materialized view per garantire la compatibilità con le versioni precedenti.

Le viste materializzate di tabelle remote richiedono query su tabelle remote. Di conseguenza, è necessario disporre di privilegi che consentano l'uso di un database link che acceda al database remoto. Il collegamento utilizzato può essere pubblico o privato. Se è privato, è necessario disporre del privilegio di sistema CREATE DATABASE LINK. Per ulteriori informazioni sui database link, si rimanda al Capitolo 22.

Se si creano le viste materializzate per sfruttare la funzionalità di *riscrittura della query* (con la quale l'ottimizzatore sceglie in modo dinamico di selezionare dei dati dalla vista materializzata invece che dalla tabella base), si deve disporre del privilegio QUERY REWRITE. Se le tabelle si trovano nello schema di un altro utente, è necessario il privilegio GLOBAL QUERY REWRITE.

## 23.3 Privilegi di tabella richiesti

Quando si crea una vista materializzata, è possibile fare riferimento alle tabelle di un database remoto tramite un database link. L'account utilizzato dal database link nel database remoto deve

avere accesso alle tabelle e alle viste utilizzate dal link stesso. Non è possibile creare una vista materializzata basata su oggetti di proprietà dell'utente SYS.

All'interno del database locale, è possibile concedere ad altri utenti locali il privilegio SELECT su una vista materializzata. Dato che la maggior parte delle viste materializzate è di sola lettura (anche se possono essere aggiornabili), non sono necessarie altre concessioni. Se si crea una vista materializzata aggiornabile, si dovrà concedere agli utenti il privilegio UPDATE sia sulla vista materializzata sia sulla tabella locale di base cui accede. Per informazioni sugli oggetti locali creati dalle viste materializzate, si rimanda al paragrafo "Oggetti locali e remoti creati" più avanti in questo capitolo.

## 23.4 Viste materializzate di sola lettura e viste materializzate aggiornabili a confronto

Una vista materializzata di sola lettura non è in grado di restituire alla sua tabella master le modifiche apportate ai dati. Una vista materializzata aggiornabile è in grado di inviare modifiche alla propria tabella master.

Anche se la distinzione può apparire semplice, le differenze basilari tra questi due tipi di viste materializzate non sono banali. Una vista materializzata di sola lettura viene implementata come un comando create table as select. Quando si verificano le transazioni, queste avvengono solo all'interno della tabella master; queste transazioni vengono in seguito inviate alla vista materializzata di sola lettura. Pertanto, il metodo in base al quale le righe della vista materializzata cambiano è controllato, le righe della vista materializzata cambiano solo in conseguenza di una modifica alla tabella master della vista materializzata.

In una vista materializzata aggiornabile, esiste meno controllo sul metodo con cui le righe della vista materializzata vengono modificate. Le righe possono essere modificate in seguito a modifiche alla tabella master o direttamente da utenti della vista materializzata. Di conseguenza, sarà necessario inviare dei record dalla tabella master alla vista materializzata e viceversa. Dato che esistono più fonti per le modifiche, esistono anche più master (indicati come *configurazione con più siti principali*).

Se si utilizzano viste materializzate aggiornabili, è necessario trattare la vista materializzata come un master, completa di tutte le strutture di replica basilari e di tutte le caratteristiche normalmente presenti nei siti principali. È inoltre necessario decidere come propagare i record dalla vista materializzata al master. Durante il trasferimento dei record dalla vista materializzata al master è necessario decidere come risolvere i conflitti. Per esempio, cosa accade se il record con ID=1 viene eliminato dal sito della vista materializzata mentre nel sito principale, in una tabella separata, viene creato un record che fa riferimento (mediante una chiave esterna) al record ID=1? Non è possibile eliminare il record ID=1 dal sito principale, poiché tale record ha un record "figlio" che vi fa riferimento. Non è possibile inserire il record figlio nel sito della vista materializzata, poiché il record padre (ID=1) è stato eliminato. Com'è possibile risolvere conflitti di questo genere?

Le viste materializzate di sola lettura consentono di evitare la necessità di risolvere conflitti, forzando il verificarsi di tutte le transazioni nella tabella master controllata. Ciò può porre dei limiti alla funzionalità, ma è una soluzione adeguata per la maggioranza delle esigenze di replica. Se si necessita della replica multimaster, è consigliabile consultare il documento *Oracle9i Replication* per avere alcune linee guida e istruzioni dettagliate sull'implementazione.

## 23.5 Sintassi di create materialized view

La sintassi base per la creazione di una vista materializzata è riportata nel listato seguente. Per la sintassi completa del comando, si rimanda al Capitolo 42. Dopo la descrizione del comando, seguono alcuni esempi che illustrano la creazione di repliche locali di dati remoti.

```
create materialized view [utente.]nome
[ organization index clausola_iot]
[ { { clausole attributi segmenti }
  | cluster cluster (colonna [. colonna] ... ) }
  [ {clausola partizionamento | clausola parallela | clausola costruzione } ]
  | on prebuilt table [ {with | without} reduced precision ] ]
[ using index
[ { clausole attributi fisici| clausola tablespace }
  [clausole attributi fisici| clausola tablespace ]
  | using no index ]
[clausola aggiornamento ]
[ for update ] [{disable | enable} query rewrite]
as subquery;
```

Il comando `create materialized view` ha quattro sezioni principali. La prima di esse è l'intestazione, in cui viene nominata la vista materializzata (la prima linea del listato):

```
create materialized view [utente.]nome
```

La vista materializzata viene creata nell'account dell'utente (schema), a meno che non venga specificato un nome di utente differente nell'intestazione. Nella seconda sezione vengono impostati i parametri di memorizzazione:

```
[ organization index clausola_iot]
[ { { clausola attributi fisici }
  | cluster cluster (colonna [. colonna] ... ) }
  [ {clausola partizionamento | clausola parallela | clausola costruzione } ]
  | on prebuilt table [ {with | without} reduced precision ] ]
[ using index
[ { clausole attributi fisici| clausola tablespace }
  [clausole attributi fisici| clausola tablespace ]
  | using no index ]
```

I parametri di memorizzazione vengono applicati a una tabella che verrà creata nel database locale. Per informazioni sui parametri di memorizzazione disponibili, si consulti la voce “Memorizzazione” nel Capitolo 42. Se i dati sono stati appena replicati in una tabella locale, è possibile utilizzare la clausola `on prebuilt table` per comunicare a Oracle di usare questa tabella come vista materializzata.

**NOTA** È possibile specificare i parametri di memorizzazione che devono essere utilizzati per l'indice che viene creato automaticamente sulla vista materializzata.

Nella terza sezione, sono impostate le opzioni di aggiornamento:

```
[ clausola aggiornamento ]
```

La sintassi per la `clausola di aggiornamento` è la seguente:

```

{ refresh
  { { fast | complete | force }
  | on { demand | commit }
  | { start with | next } data
  | with { primary key | rowid }
  | using
    { default [ master | local ] rollback segment
    | [ master | local ] rollback segment segmento_rollback
    }
  [ default [ master | local ] rollback segment
  | [ master | local ] rollback segment segmento_rollback
  ]...
}

[ { fast | complete | force }
| on { demand | commit }
| { start with | next } data
| with { primary key | rowid }
| using
  { default [ master | local ] rollback segment
  | [ master | local ] rollback segment segmento_rollback
  }
  [ default [ master | local ] rollback segment
  | [ master | local ] rollback segment segmento_rollback
  ]...
]

]...
| never refresh
}

```

L'opzione `refresh` specifica il meccanismo che Oracle deve usare per l'aggiornamento della vista materializzata. Le tre opzioni disponibili sono: `fast`, `complete` e `force`. Gli aggiornamenti rapidi sono disponibili solo se Oracle è in grado di associare le righe nella vista materializzata direttamente alle righe delle tabelle base; questi aggiornamenti usano tabelle denominate *log delle viste materializzate* per inviare righe specifiche dalla tabella master alla vista materializzata. Gli aggiornamenti completi ricreano completamente la vista materializzata. L'opzione `force` per gli aggiornamenti completi comunica a Oracle di utilizzare un aggiornamento rapido se disponibile, altrimenti un aggiornamento completo. Se si è creata una vista materializzata semplice, ma si desidera eseguire aggiornamenti completi, è necessario specificare `refresh complete` nel comando `create materialized view`. Le opzioni di aggiornamento sono descritte in modo più dettagliato nel paragrafo “Aggiornamento delle viste materializzate”, più avanti nel capitolo. All'interno di questa sezione del comando `create materialized view` è anche possibile specificare il meccanismo utilizzato per correlare i valori nella vista materializzata a quelli della tabella master, ovvero stabilire se devono essere utilizzati valori RowID o valori della chiave primaria. Per default, vengono utilizzate le chiavi primarie.

Se la query master per la vista materializzata fa riferimento a un join o ad un aggregato composto da un'unica tabella, è possibile usare l'opzione `on commit` per controllare la replica delle modifiche. Con questa opzione, le modifiche verranno inviate dal master alla replica mentre vengono trasmesse alla tabella master. Se invece si specifica l'opzione `on demand`, l'aggiornamento avviene quando si esegue manualmente un comando di aggiornamento.

La quarta sezione del comando `create materialized view` contiene la query utilizzata dalla vista materializzata:

```
[ for update ] [{disable | enable} query rewrite]
as subquery
```

Se si specifica `FOR UPDATE`, la vista materializzata risulta aggiornabile; in caso contrario è di sola lettura. La maggior parte delle viste materializzate è una replica di sola lettura dei dati master. Se si utilizzano viste materializzate aggiornabili, occorre essere consapevoli di possibili problemi come la replica a due vie delle modifiche e la risoluzione di conflitti nelle modifiche ai dati. Le viste materializzate aggiornabili sono un esempio di replica multimaster; per ulteriori dettagli sull'implementazione di un ambiente di replica multimaster, si consulti la guida *Oracle9i Replication*.

**NOTA** *La query che forma la base per la vista materializzata non deve utilizzare le pseudocolonne Utente o SysDate.*

Nell'esempio seguente, viene creata in un database locale una vista materializzata di sola lettura denominata `BIBLIOTECA_LOCALE`, basata su una tabella remota `BIBLIOTECA`, accessibile tramite il database link `REMOTE_CONNECT`. La vista materializzata viene inserita nella tablespace `UTENTI`.

```
create materialized view BIBLIOTECA_LOCALE
storage (initial 100K next 100K pctincrease 0)
tablespace UTENTI
refresh force
start with SysDate next SysDate+7
with primary key
as
select * from BIBLIOTECA@REMOTE_CONNECT;
```

Oracle risponde con:

Materialized view created.

Il comando dell'esempio precedente creerà una vista materializzata di sola lettura di nome `BIBLIOTECA_LOCALE`. La sua tabella base verrà creata con i parametri di memorizzazione specificati in una tablespace di nome `UTENTI`. Dal momento che i dati contenuti nella tabella base locale della vista materializzata cambieranno nel tempo, sarebbe opportuno memorizzare nei database di produzione le viste materializzate nelle proprie tablespace. L'opzione di aggiornamento `force` viene specificata poiché per la vista materializzata non esiste un log delle viste materializzate nella tabella base; Oracle cercherà di eseguire un aggiornamento rapido ma alla fine ricorrerà a un aggiornamento completo fino a quando il log delle viste materializzate non verrà creato. La query della vista materializzata specifica che l'intera tabella `BIBLIOTECA`, senza modifiche, dovrà essere copiata nel database locale. Non appena la vista materializzata `BIBLIOTECA_LOCALE` è stata creata, la sua tabella base verrà popolata con i dati della tabella `BIBLIOTECA`. Da quel momento in poi la vista materializzata verrà aggiornata ogni sette giorni. I parametri di memorizzazione che non sono specificati utilizzano i valori predefiniti in relazione alla tablespace `UTENTI`.

Nell'esempio seguente, viene creata in un database locale una vista materializzata di nome `CONTA_CATEGORIA_LOCALE`, basata su una tabella remota di nome `BIBLIOTECA` in un database cui si accede tramite il database link `REMOTE_CONNECT`.

```
create materialized view CONTA_CATEGORIA_LOCALE
storage (initial 50K next 50K pctincrease 0)
tablespace UTENTI
refresh force
start with SysDate next SysDate+7
as
```

---

```
select NomeCategoria, COUNT(*) CountPerCat
  from BIBLIOTECA@REMOTE_CONNECT
 group by NomeCategoria;
```

La query eseguita sulla vista materializzata CONTA\_CATEGORIA\_LOCALE conta il numero di libri di ogni categoria nella tabella remota BIBLIOTECA.

I due esempi di questo paragrafo presentano numerosi aspetti importanti:

- La query group by utilizzata sulla vista materializzata CONTA\_CATEGORIA\_LOCALE potrebbe essere eseguita in SQLPLUS sulla vista materializzata BIBLIOTECA\_LOCALE. Ciò significa che l'operazione group by può essere eseguita al di fuori della vista materializzata.
- Dato che CONTA\_CATEGORIA\_LOCALE usa una clausola group by, si tratta di una vista materializzata complessa, e potrebbe essere in grado di eseguire solo aggiornamenti completi. Dal momento che BIBLIOTECA\_LOCALE è una vista materializzata semplice, può eseguire aggiornamenti rapidi.

Le due viste materializzate descritte negli esempi precedenti fanno riferimento alla stessa tabella. Dato che una di queste due è una semplice vista materializzata che replica tutte le colonne e le righe della tabella master, la seconda può apparire a prima vista ridondante. Tuttavia, ogni tanto la seconda vista materializzata, quella complessa, è la più utile delle due.

Come è possibile? Per prima cosa, occorre ricordare che queste viste materializzate vengono utilizzate per servire le esigenze di query degli utenti *locali*. Se questi utenti eseguono sempre operazioni group by nelle query, e le colonne di raggruppamento sono fisse, allora per essi può rivelarsi più utile CONTA\_CATEGORIA\_LOCALE. In secondo luogo, se il volume di transazioni sulla tabella master BIBLIOTECA è ingente, o se questa tabella è di dimensioni molto ridotte, potranno non esserci differenze significative nei tempi di aggiornamento degli aggiornamenti rapidi e di quelli completi. La vista materializzata più appropriata è quella che si dimostra più produttiva per gli utenti.

## Viste materializzate basate su RowID e viste materializzate basate su chiave primaria

Invece di basare la viste materializzate sui valori di RowID della tabella master, è possibile basarle sui valori di chiave primaria della tabella master. Per decidere quale opzione scegliere, è necessario considerare più fattori:

- **Stabilità del sistema** Se il sito principale non è stabile, potrebbe presentarsi la necessità di eseguire recuperi del database che coinvolgono la tabella master. Quando si usano le utility Import e Export di Oracle per effettuare i recuperi, è molto probabile che i valori ROWID delle righe cambino. Se il sistema richiede frequenti operazioni di esportazione e importazione, è consigliato l'uso di viste materializzate basate su valori di chiave primaria.
- **Quantità di dati replicati** Se normalmente non si replicano le colonne di chiave primaria, è possibile ridurre la quantità dei dati replicati replicando i valori di ROWID al posto di quelli di chiave primaria.
- **Quantità di dati inviati durante gli aggiornamenti** Durante gli aggiornamenti, le viste materializzate basate sui valori ROWID in genere richiedono di inviare alla vista materializzata una quantità di dati minore rispetto alle viste materializzate basate su valori di chiave primaria (a meno che la chiave primaria sia una colonna molto breve).
- **Dimensioni della tabella log delle viste materializzate** Oracle consente di memorizzare le modifiche apportate alle tabelle master in tabelle separate, dette *log delle viste materializzate* (descritte più avanti in questo capitolo). Se la chiave primaria è costituita da più colonne, la tabella log delle viste materializzate per una vista materializzata basata sui valori di chiave pri-

maria potrà essere significativamente più grande del log delle viste materializzate per una vista materializzata analoga basata su valori di ROWID.

- **Integrità referenziale** Per utilizzare le viste materializzate basate su valori di chiave primaria, è necessario definire una chiave primaria nella tabella master. Se non è possibile definire una chiave primaria nella tabella master, allora si dovrà ripiegare su viste materializzate basate su valori ROWID.

## Oggetti di base creati

Quando si crea una vista materializzata, nel database locale e in quello remoto viene creato un certo numero di oggetti. Gli oggetti di supporto creati all'interno di un database sono gli stessi sia per le viste materializzate semplici sia per quelle complesse. Con le viste materializzate semplici, si ha la possibilità di creare oggetti aggiuntivi chiamati *log delle viste materializzate*, che verranno analizzati nel paragrafo “Sintassi di create materialized view log,” più avanti nel capitolo.

Si consideri la vista materializzata semplice descritta nell'ultimo paragrafo:

```
create materialized view BIBLIOTECA_LOCALE
storage (initial 100K next 100K pctincrease 0)
tablespace USERS
refresh force
start with SysDate next SysDate+7
with primary key
as
select * from BIBLIOTECA@REMOTE_CONNECT;
```

All'interno del database locale, questo comando creerà gli oggetti seguenti nello schema del proprietario della vista materializzata oltre agli oggetti della vista materializzata stessa:

- Una tabella di nome BIBLIOTECA\_LOCALE che è la *tabella base locale* per la vista materializzata della tabella remota. Questa tabella contiene i dati replicati.
- Un indice sulla tabella base locale (BIBLIOTECA\_LOCALE) della vista materializzata (fare riferimento al prossimo paragrafo “Indicizzazione delle tabelle delle viste materializzate”). L'indice di questo esempio è creato sulla colonna Titolo, la chiave primaria della tabella di origine per la vista materializzata.

È ammessa una sola modifica agli oggetti di base: la tabella BIBLIOTECA\_LOCALE dovrebbe essere indicizzata in modo da rispecchiare i percorsi di query normalmente utilizzati dagli utenti locali. Quando si indicizza la tabella base locale della vista materializzata, è necessario considerare i requisiti di memorizzazione degli indici mentre si valutano le esigenze di spazio della vista materializzata. Per ulteriori informazioni, si rimanda al prossimo paragrafo, “Indicizzazione delle tabelle delle viste materializzate”.

Nel database remoto non vengono creati oggetti di supporto, a meno che non si utilizzino log delle viste materializzate per registrare le modifiche apportate alle righe della tabella master. I log delle viste materializzate sono descritti nel paragrafo “Sintassi di create materialized view log”, più avanti nel capitolo.

## Indicizzazione delle tabelle delle viste materializzate

Come emerso dalla discussione precedente, la tabella base locale contiene i dati replicati. Dato che tali dati sono stati replicati con uno scopo preciso (solitamente il miglioramento delle prestazioni del database o della rete), è importante proseguire mantenendo questo scopo anche dopo la

creazione della vista materializzata. In genere, i miglioramenti nelle prestazioni delle query sono ottenuti grazie all'uso degli indici. Le colonne che vengono utilizzate di frequente nelle clausole `where` delle query dovrebbero essere indicizzate. Se nelle query si accede spesso a un gruppo di colonne, si potrà creare su di esse un indice concatenato (per ulteriori informazioni sull'ottimizzatore di Oracle, si rimanda al capitolo 38).

Oracle non crea automaticamente degli indici per le viste materializzate complesse su colonne diverse da quelle di chiave primaria. Questi indici devono essere creati manualmente. Per crearli sulla tabella base locale, è necessario ricorrere al comando `create index` (si consulti il Capitolo 42). *Non* si devono creare vincoli sulla tabella base locale della vista materializzata.

Dal momento che sulle colonne su cui gli utenti probabilmente eseguiranno query dalla vista materializzata non vengono creati indici, questi dovrebbero essere creati sulla tabella base locale della vista materializzata.

## **Uso di viste materializzate per alterare i percorsi di esecuzione delle query**

Per un database di grandi dimensioni, una vista materializzata può offrire diversi vantaggi nelle prestazioni. Le viste materializzate possono essere utilizzate per indurre l'ottimizzatore a modificare i percorsi di esecuzione delle query. Questa caratteristica, denominata *riscrittura delle query*, consente all'ottimizzatore di utilizzare una vista materializzata al posto della tabella su cui la vista materializzata stessa ha eseguito una query, anche se quest'ultima non è citata nella query. Per esempio, se si ha una tabella `VENDITE` di grandi dimensioni, è possibile creare una vista materializzata che somma i dati di `VENDITE` in base alle regioni. Se un utente esegue una query sulla tabella `VENDITE` per ottenere la somma dei dati di `VENDITE` relativi a una determinata regione, Oracle potrà reindirizzare questa query in modo che utilizzi la vista materializzata al posto della tabella `VENDITE`. Di conseguenza, si potrà ridurre il numero di accessi alle tabelle più voluminose, migliorando le prestazioni del sistema. Inoltre, dal momento che i dati della vista materializzata sono già suddivisi per regione, non sarà necessario eseguire la somma mentre si effettua la query.

**NOTA** È necessario specificare query rewrite nella definizione della vista materializzata per far sì che la vista venga utilizzata nell'operazione di riscrittura della query.

Per usare la caratteristica di riscrittura delle query in modo efficiente, si dovrebbe creare una dimensione che definisce le gerarchie all'interno dei dati della tabella. Per esempio, i Paesi fanno parte dei continenti, quindi si possono creare delle tabelle per supportare questa gerarchia:

```
create dimension GEOGRAFIA
  level PAESE_ID      is PAESE.Paese
  level CONTINENTE_Id is CONTINENTE.Continent
  hierarchy PAESE_ROLLUP (
    PAESE_ID            child of
    CONTINENTE_ID
    join key PAESE.Continent references CONTINENTE_id);
```

Se si sommano i dati di `VENDITE` in una vista materializzata a livello di paese, l'ottimizzatore sarà in grado di reindirizzare le query per i dati di `VENDITE` a livello di paese alla vista materializzata. Dato che la vista materializzata dovrebbe contenere meno dati della tabella `VENDITE`, la query sulla vista materializzata dovrebbe produrre un miglioramento delle prestazioni rispetto a una query analoga eseguita sulla tabella `VENDITE`.

Per abilitare una vista materializzata alla riscrittura delle query, tutte le tabelle master della vista materializzata dovranno trovarsi nello schema della vista materializzata e si dovrà disporre del privilegio di sistema QUERY REWRITE. Se la vista e le tabelle si trovano in schemi separati, allora si dovrà disporre del privilegio di sistema GLOBAL QUERY REWRITE. In linea di massima, le viste materializzate andrebbero create nello stesso schema delle tabelle su cui si basano; in caso contrario sarà necessario gestire i permessi e le concessioni necessari per creare e gestire la vista materializzata.

## 23.6 Aggiornamento delle viste materializzate

I dati contenuti in una vista materializzata potrebbero essere replicati in un'unica volta (quando si crea la vista) oppure a intervalli specifici. Il comando create materialized view consente di impostare l'intervallo di aggiornamento, delegando al database la responsabilità di programmare ed eseguire questi aggiornamenti. Nei paragrafi successivi viene descritta l'esecuzione di aggiornamenti manuali e automatici.

### Quale aggiornamento è possibile eseguire?

Per visualizzare le possibilità di riscrittura e aggiornamento a disposizione delle viste materializzate, è possibile eseguire una query sulla tabella TABELLA\_CAPACITA\_VM. Da una versione all'altra queste possibilità possono cambiare, quindi sarebbe opportuno rivalutarle dopo ogni aggiornamento del software Oracle. Per creare questa tabella, si esegua lo script utlxmv.sql situato nella directory /rdbms/admin sotto la directory home del software Oracle.

Le colonne di TABELLA\_CAPACITA\_VM sono

Nome	Null?	Type
ISTRUZIONE_ID		VARCHAR2(30)
PROPRIETARIOVM		VARCHAR2(30)
NOMEVM		VARCHAR2(30)
NOME_CAPACITA		VARCHAR2(30)
POSSIBILE		CHAR(1)
TESTO_CORRELATO		VARCHAR2(2000)
NUME_CORRELATO		NUMBER
MSGNO		NUMBER(38)
MSGTXT		VARCHAR2(2000)
SEQ		NUMBER

Per popolare la tabella TABELLA\_CAPACITA\_VM, è necessario eseguire la procedura DBMS\_MVIEW.EXPLAIN\_MVIEW, usando il nome della vista materializzata come valore di input, come mostrato nel listato seguente.

```
execute DBMS_MVIEW.EXPLAIN_MVIEW('biblioteca_locale');
```

Lo script utlxmv.sql aiuta nell'interpretazione dei valori di colonna, come mostrato nel listato seguente.



```

TESTO_CORRELATO VARCHAR(2000),-- Proprietario.tabella.colonna,
-- alias nome, etc.
-- correlato a questo messaggio.
-- Il significato specifico di questa
-- colonna dipende dalla colonna MSGNO.
-- Si consulti la documentazione per
-- DBMS_MVIEW.EXPLAIN_MVIEW()
-- per i dettagli
NUM_CORRELATO NUMBER, -- quando c'è un valore numerico associato
-- a una riga, va inserito qui.
-- Il significato specifico di
-- questa colonna
-- dipende dalla colonna MSGNO. Si consulti
-- la documentazione
-- DBMS_MVIEW.EXPLAIN_MVIEW()
-- per i dettagli
MSGNO INTEGER, -- quando disponibile, messaggio QSM #
-- che spiega come mai non è possibile
-- oppure fornisce più dettagli
-- quando abilitato.
MSGTXT VARCHAR(2000),-- Testo associato a MSGNO.
SEQ NUMBER); -- Utile nella clausola ORDER BY quando
-- si eseguono selezioni da questa tabella.

```

Terminata l'esecuzione della procedura EXPLAIN\_MVIEW, si potrà eseguire una query su TABELLA\_CAPACITA\_VM per stabilire le opzioni disponibili.

```

select Nome_Capacita, Msgtxt
  from TABELLA_CAPACITA_VM
 where Msgtxt is not null;

```

Per la vista materializzata BIBLIOTECA\_LOCALE, la query restituisce:

```

NOME_CAPACITA
-----
MSGTXT
-----
TABELLA_PCT
la relazione non è una tabella partizionata

REFRESH_FAST_AFTER_INSERT
la tabella dei dettagli non contiene un log delle viste materializzate

REFRESH_FAST_AFTER_ONETAB_DML
vedere perché REFRESH_FAST_AFTER_INSERT è disattivato

REFRESH_FAST_AFTER_ANY_DML
vedere perché REFRESH_FAST_AFTER_ONETAB_DML è disattivato

REFRESH_FAST_PCT
PCT non può essere eseguito su nessuna delle tabelle di dettagli nella vista materializzata

REWRITE_FULL_TEXT_MATCH
la riscrittura delle query è disattivata nella vista materializzata
REWRITE_PARTIAL_TEXT_MATCH
la riscrittura delle query è disattivata nella vista materializzata

```

**REWRITE\_GENERAL**

riscrittura delle query è disattivata nella vista materializzata

**REWRITE\_PCT**

la riscrittura generale non è consentita e PCT non può essere eseguito su nessuna delle tabelle di dettagli

Dato che la clausola `query rewrite` non è stata specificata durante la creazione della vista materializzata, le capacità di riscrittura delle query non sono state attivate per `BIBLIOTECA_LOCALE`. Le capacità di aggiornamento rapido non sono supportate perché la tabella base non ha un log delle viste materializzate. Se si modifica la vista materializzata o la sua tabella base, si dovrebbero creare nuovamente i dati di `TABELLA_CAPACITA_VM` per visualizzare le nuove capacità.

## Aggiornamenti automatici

Si consideri la vista materializzata `BIBLIOTECA_LOCALE` descritta in precedenza. Le impostazioni relative agli intervalli di aggiornamento, definite dal comando `create materialized view`, sono riportate in grassetto nel listato seguente:

```
create materialized view BIBLIOTECA_LOCALE
storage (initial 100K next 100K pctincrease 0)
tablespace USERS
refresh force
start with SysDate next SysDate+7
with primary key
as
select * from BIBLIOTECA@REMOTE_CONNECT;
```

La programmazione degli aggiornamenti ha tre componenti. In primo luogo, viene specificato il tipo di aggiornamento (`fast`, `complete` o `force`). Gli aggiornamenti rapidi utilizzano *log delle viste materializzate* (descritti più avanti in questo capitolo) per inviare le righe modificate dalla tabella master alla vista materializzata. Gli aggiornamenti completi ricreano completamente la vista materializzata. L'opzione `force` per gli aggiornamenti indica a Oracle di usare, se disponibile, un aggiornamento rapido; altrimenti un aggiornamento completo.

La clausola `start with` specifica al database quando effettuare la prima replica dalla tabella master alla tabella base locale. Essa deve effettuare una stima relativa a un istante futuro. Se non si specifica un tempo `start with` ma un valore `next`, Oracle utilizza la clausola `next` per determinare il tempo d'inizio. Per mantenere il controllo sugli intervalli di replica, è necessario specificare un valore per la clausola `start with`.

La clausola `next` indica a Oracle quanto dovrà attendere tra un aggiornamento e quello successivo. Dato che viene applicata a un orario base differente, ogni volta che la vista materializzata è aggiornata, la clausola `next` specifica un'espressione di data al posto di una data fissa. Nell'esempio precedente l'espressione è

`next SysDate+7`

Ogni volta che si aggiorna la vista materializzata, l'aggiornamento successivo verrà programmato sette giorni dopo. Anche se in questo esempio la programmazione dell'aggiornamento è piuttosto semplice, è possibile utilizzare numerose funzioni di data di Oracle per personalizzare la pianificazione degli aggiornamenti. Per esempio, se si desidera eseguire un aggiornamento ogni lunedì a mezzogiorno, a prescindere dalla data corrente, si può impostare la clausola `next` a

---

```
NEXT_DAY(TRUNC(SysDate), 'MONDAY')+12/24
```

Così si individua il primo lunedì dopo la data corrente di sistema. La porzione relativa all'ora di tale data viene troncata e vengono aggiunte 12 ore alla data. Per informazioni sulle funzioni di data in Oracle, si rimanda al Capitolo 9.

Per abilitare gli aggiornamenti automatici delle viste materializzate, occorre fare in modo che nel database venga eseguito almeno un processo di background per l'aggiornamento delle viste materializzate. Il procedimento di aggiornamento, detto *Jnnn* (dove *nnn* è un numero compreso tra 000 e 999) si attiva periodicamente e verifica se qualche vista materializzata del database dev'essere aggiornata. Il numero di processi *Jnnn* eseguiti nel database è determinato da un parametro di inizializzazione denominato JOB\_QUEUE\_PROCESSES. Questo parametro dev'essere impostato (nel file dei parametri di inizializzazione) a un valore maggiore di 0. Nella maggior parte dei casi è sufficiente un valore pari a 1. Un processo di coordinamento avvia i processi di job queue in base alle esigenze.

Se il database non esegue i processi di *Jnnn*, si dovrà ricorrere a metodi di aggiornamento manuali, descritti nel prossimo paragrafo.

## Aggiornamenti manuali

Oltre agli aggiornamenti automatici del database, è anche possibile eseguire aggiornamenti manuali delle viste materializzate. Questi sostituiscono gli aggiornamenti programmati normalmente. Il nuovo valore di start with si baserà sull'ora dell'aggiornamento manuale.

Per aggiornare un'unica vista materializzata, si utilizzi DBMS\_MVIEW.REFRESH. I due parametri principali di questa procedura sono il nome della vista materializzata da aggiornare e il metodo da utilizzare. Per questo metodo, si può specificare "c" per un aggiornamento completo, "r" per un aggiornamento rapido e "?" per la clausola force. Per esempio:

```
execute DBMS_MVIEW.REFRESH('biblioteca_locale', 'c');
```

Se con un'unica esecuzione di DBMS\_MVIEW.REFRESH si aggiornano più viste materializzate, si dovranno elencare i nomi di tutte le viste materializzate nel primo parametro, nonché i metodi di aggiornamento corrispondenti nel secondo parametro, come mostrato di seguito:

```
execute DBMS_MVIEW.REFRESH('biblioteca_locale', 'conta_categoria_locale', '?c');
```

In questo esempio, la vista materializzata denominata BIBLIOTECA\_LOCALE viene aggiornata con un aggiornamento rapido se possibile (con la clausola force), mentre la seconda vista materializzata eseguirà un aggiornamento completo.

Per aggiornare tutte le viste materializzate per le quali è previsto un aggiornamento automatico, si può utilizzare una procedura separata nel package DBMS\_MVIEW. Questa procedura, il cui nome è REFRESH\_ALL, aggiornerà ciascuna vista materializzata separatamente. La procedura non accetta parametri. Ecco un esempio della sua esecuzione:

```
execute DBMS_MVIEW.REFRESH_ALL;
```

Dato che le viste materializzate verranno aggiornate tramite REFRESH\_ALL una di seguito all'altra, non potranno essere aggiornate tutte alla stessa ora. Di conseguenza, un guasto al server o al database durante l'esecuzione di questa procedura potrà provocare la perdita di sincronizzazione a livello locale tra una vista materializzata e l'altra. In tal caso sarà sufficiente avviare

nuovamente la procedura, una volta recuperato il database. In alternativa, si possono creare gruppi di aggiornamento, come descritto nel prossimo paragrafo.

Un'altra procedura all'interno di DBMS\_MVIEWS, REFRESH\_ALL\_MVIEWS, aggiorna tutte le viste materializzate che hanno le proprietà seguenti:

- La vista materializzata non è stata aggiornata dall'ultima modifica apportata alla tabella master o alla vista materializzata master da cui dipende.
- La vista materializzata e tutte le tabelle master o le viste materializzate master da cui dipende sono locali.
- La vista materializzata si trova nella vista DBA\_MVIEWS.

### **Imposizione dell'integrità referenziale tra le viste materializzate**

L'integrità referenziale tra due tabelle correlate, sulle quali si basano viste materializzate semplici, potrebbe non essere imposta nelle rispettive viste materializzate. Se le due tabelle vengono aggiornate a orari diversi, oppure se durante l'aggiornamento si verificano delle transazioni sulle tabelle master, può accadere che le viste materializzate di queste tabelle non riflettano l'integrità referenziale delle tabelle master.

Per esempio, le tabelle BIBLIOTECA e BIBLIOTECA\_PRESTITO sono correlate tra loro tramite una relazione chiave primaria/chiave esterna, pertanto le viste materializzate semplici di queste tabelle possono contenere violazioni a questa relazione, per esempio includendo chiavi esterne senza le corrispondenti chiavi primarie. In questo caso, ciò potrebbe significare la presenza nella vista materializzata BIBLIOTECA\_PRESTITO di record con valori della colonna Titolo che non esistono nella vista materializzata BIBLIOTECA.

Esistono numerose soluzioni possibili a questo problema. La prima consiste nel programmare gli aggiornamenti in modo che avvengano quando le tabelle master non sono utilizzate. In secondo luogo è possibile eseguire gli aggiornamenti manualmente (per informazioni su questo argomento si consulti il prossimo paragrafo) subito dopo aver bloccato le tabelle master o aver reso quiescente il database. In terzo luogo si potrebbero unire le tabelle tramite join nella vista materializzata, creando una vista materializzata complessa che si baserà sulle tabelle master (correlate le une alle altre in modo adeguato).

L'uso di gruppi di aggiornamento è la quarta soluzione possibile al problema legato all'integrità referenziale. È possibile raggruppare le viste materializzate correlate in *gruppi di aggiornamento*. Lo scopo di un gruppo di aggiornamento è coordinare gli intervalli di aggiornamento dei propri membri. Le viste materializzate le cui tabelle master hanno relazioni con altre tabelle master sono ottime candidate a far parte di gruppi di aggiornamento. Il coordinamento degli intervalli di aggiornamento delle viste materializzate conserverà anche in queste ultime l'integrità referenziale delle tabelle master. Se non si utilizzano i gruppi di aggiornamento, i dati contenuti nelle viste materializzate potranno risultare incoerenti rispetto all'integrità referenziale delle tabelle master.

La manipolazione dei gruppi di aggiornamento viene eseguita tramite il package DBMS\_REFRESH. Le procedure contenute in questo package sono MAKE, ADD, SUBTRACT, CHANGE, DESTROY e REFRESH, come mostrato negli esempi seguenti. Le informazioni sui gruppi di aggiornamento già esistenti possono essere ottenute tramite query sulle viste del dizionario dati USER\_REFRESH e USER\_REFRESH\_CHILDREN.

**NOTA** *Le viste materializzate appartenenti a un gruppo di aggiornamento non devono obbligatoriamente appartenere allo stesso schema, ma devono essere memorizzate tutte all'interno dello stesso database.*

Si crei un gruppo di aggiornamento eseguendo la procedura MAKE del package DBMS\_REFRESH. La struttura di tale procedura è riportata nel listato seguente:

```
DBMS_REFRESH.MAKE
(name IN VARCHAR2,
list IN VARCHAR2, |
tab IN DBMS_UTILITY.UNCL_ARRAY,
next_date IN DATE,
interval IN VARCHAR2,
implicit_destroy IN BOOLEAN := FALSE,
lax IN BOOLEAN := FALSE,
job IN BINARY_INTEGER := 0,
rollback_seg IN VARCHAR2 := NULL,
push_deferred_rpc IN BOOLEAN := TRUE,
refresh_after_errors IN BOOLEAN := FALSE,
purge_option IN BINARY_INTEGER := NULL,
parallelism IN BINARY_INTEGER := NULL,
heap_size IN BINARY_INTEGER := NULL);
```

Tutti i parametri di questa procedura, esclusi i primi quattro, hanno valori predefiniti normalmente accettabili. I parametri *elenco* e *scheda* si escludono a vicenda. Per creare un gruppo di aggiornamento per le viste materializzate BIBLIOTECA\_LOCALE e CONTA\_CATEGORIA\_LOCALE, si può usare il comando seguente. Il comando viene mostrato separato su più linee, con caratteri di continuazione alla fine di ogni linea; tuttavia, lo si può inserire anche in un'unica linea.

```
execute DBMS_REFRESH.MAKE -
(name => 'gruppo_libro', -
list => 'biblioteca_locale, conta_categoria_locale', -
next_date => SysDate, -
interval => 'SysDate+7');
```

**NOTA** Il parametro *list*, che è il secondo nel listato, ha un apice all'inizio e uno alla fine, senza altri apici intermedi. In questo esempio, due viste materializzate, BIBLIOTECA\_LOCALE e CONTA\_CATEGORIA\_LOCALE, sono state passate alla procedura tramite un singolo parametro.

Il comando precedente creerà un gruppo di aggiornamento di nome GRUPPO\_LIBRO, che ha due viste materializzate come membri. Il nome del gruppo di aggiornamento è racchiuso tra apici, come il parametro *list* dei membri, lo stesso non vale però per ogni membro.

Se il gruppo di aggiornamento contiene una vista materializzata che è già membro di un altro gruppo di aggiornamento (per esempio per lo spostamento di una vista materializzata da un vecchio gruppo di aggiornamento a uno appena creato), è necessario impostare il parametro *lax* a TRUE, in modo che Oracle rimuova automaticamente questa vista materializzata dal gruppo di aggiornamento vecchio. Una vista materializzata può anche appartenere a un solo gruppo di aggiornamento per volta.

Per aggiungere viste materializzate a un gruppo di aggiornamento già esistente, si utilizzi la procedura ADD del package DBMS\_REFRESH, la cui struttura è

```
DBMS_REFRESH.ADD
(name IN VARCHAR2,
list IN VARCHAR2, |
tab IN DBMS_UTILITY.UNCL_ARRAY,
lax IN BOOLEAN := FALSE);
```

Come avviene per la procedura MAKE, il parametro *lax* della procedura ADD non dev'essere necessariamente specificato, a meno che una vista materializzata non venga spostata da un gruppo di aggiornamento a un altro. Quando questa procedura viene eseguita con il parametro *lax* impostato su TRUE, la vista materializzata viene spostata nel nuovo gruppo di aggiornamento e viene automaticamente eliminata da quello vecchio.

Per rimuovere le viste materializzate da un gruppo di aggiornamento già esistente, si utilizzi la procedura SUBTRACT del package DBMS\_REFRESH, come mostrato di seguito:

```
DBMS_REFRESH.SUBTRACT
(name IN VARCHAR2,
list IN VARCHAR2. |
tab IN DBMS_UTILITY.UNCL_ARRAY,
lax IN BOOLEAN := FALSE);
```

Come avviene per le procedure MAKE e ADD, una singola vista materializzata o un elenco di viste materializzate (separate da virgolette) potrebbero fungere da input per la procedura SUBTRACT. Tramite la procedura CHANGE del package DBMS\_REFRESH è possibile modificare l'intervallo di aggiornamento di un gruppo di aggiornamento.

```
DBMS_REFRESH.CHANGE
(name IN VARCHAR2,
next_date IN DATE := NULL,
interval IN VARCHAR2 := NULL,
implicit_destroy IN BOOLEAN := NULL,
rollback_seg IN VARCHAR2 := NULL,
push_deferred_rpc IN BOOLEAN := NULL,
refresh_after_errors IN BOOLEAN := NULL,
purge_option IN BINARY_INTEGER := NULL,
parallelism IN BINARY_INTEGER := NULL,
heap_size IN BINARY_INTEGER := NULL);
```

Il parametro *next\_date* è analogo alla clausola start with nel comando create materialized view. Il parametro *interval* è analogo alla clausola next nel comando create materialized view. Per esempio, per modificare l'intervallo di aggiornamento del gruppo GRUPPO\_LIBRO in modo che venga replicato ogni tre giorni, è possibile eseguire il comando seguente (che specifica un valore NULL per il parametro *next\_date*, lasciando tale valore invariato):

```
execute DBMS_REFRESH.CHANGE
(name => 'book_group',
next_date => null,
interval => 'SysDate+3');
```

Eseguito questo comando, il ciclo di aggiornamento per il gruppo GRUPPO\_LIBRO verrà modificato in modo da aggiornare il gruppo ogni tre giorni.

**NOTA** *Le operazioni di aggiornamento sui gruppi di aggiornamento possono richiedere più tempo degli analoghi aggiornamenti di viste materializzate. Gli aggiornamenti dei gruppi possono richiedere anche uno spazio significativo per i segmenti di undo per mantenere la coerenza dei dati durante l'aggiornamento.*

Si potrebbe aggiornare manualmente un gruppo di aggiornamento tramite la procedura REFRESH del package DBMS\_REFRESH. La procedura REFRESH accetta come unico parametro il nome del gruppo di aggiornamento. Il comando riportato nel listato seguente aggiorna il gruppo di aggiornamento GRUPPO\_LIBRO:

---

```
execute DBMS_REFRESH.REFRESH('gruppo_libro');
```

Per eliminare un gruppo di aggiornamento, si utilizzi la procedura DESTROY del package DBMS\_REFRESH, come mostrato nell'esempio seguente. L'unico parametro della procedura è il nome del gruppo di aggiornamento.

```
execute DBMS_REFRESH.DESTROY(name => 'gruppo_libro');
```

Inoltre, si può impostare l'eliminazione implicita del gruppo di aggiornamento. Se si imposta il parametro *implicit\_destroy* a TRUE quando si crea il gruppo tramite la procedura MAKE, il gruppo di aggiornamento verrà eliminato (distrutto) non appena l'ultimo membro sarà stato rimosso dal gruppo stesso (solitamente con la procedura SUBTRACT).

### **Altre opzioni per la gestione delle viste materializzate**

Esistono altri due package che possono essere utilizzati per gestire e valutare le viste materializzate: DBMS\_MVIEW e DBMS OLAP. Per creare questi package per le viste materializzate, si devono eseguire rispettivamente dbmssnap.sql e dbmssqlsum.sql.

Le opzioni del package DBMS\_MVIEW sono mostrate nella Tabella 23.1.

Il package DBMS\_MVIEW serve per eseguire azioni di gestione come la valutazione, la registrazione o l'aggiornamento di una vista materializzata.

Il package DBMS OLAP può essere utilizzato per stabilire se una vista materializzata è in grado o meno di migliorare le prestazioni delle query del database, per generare script per la creazione di viste materializzate, per valutare la dimensione di una vista materializzata e così via. Il package DMBS OLAP possiede le opzioni mostrate nella Tabella 23.2.

Per ulteriori informazioni su questo package, si consulti la guida *Oracle9i Supplied PL/SQL Packages and Types Reference*.

## **23.7 Sintassi di create materialized view log**

Nelle viste materializzate semplici, ciascun record è basato su una singola riga di una singola tabella master. Quando si usano le viste materializzate semplici, è possibile creare un *log delle viste materializzate* sulla tabella master. Il log delle viste materializzate è una tabella che registra la data in cui è stata replicata per l'ultima volta ciascuna riga modificata nella tabella master. Il record delle righe modificate potrà essere utilizzato durante gli aggiornamenti per inviare alle viste materializzate soltanto quelle righe della tabella master che sono state modificate. Più viste materializzate semplici basate sulla stessa tabella possono utilizzare lo stesso log delle viste materializzate.

La sintassi completa del comando create materialized view log è mostrata nel Capitolo 42. Il listato seguente mostra una parte della sintassi; come si può notare dalla sintassi, questo comando possiede tutti i parametri che normalmente sono associati alle tabelle:

```
create materialized view log on [schema .] tabella
[ { clausola_attributi_fisici
| tablespace tablespace
| { logging | nologging }
| { cache | nocache }
}
[clausola_attributi_fisici
| tablespace tablespace
| { logging | nologging }
```

**Tabella 23.1** Sottoprogrammi di DBMS\_MVIEW.

SOTTOPROGRAMMA	DESCRIZIONE
BEGIN_TABLE_REORGANIZATION	Per preservare i dati necessari per una vista materializzata viene messo in atto un processo utilizzato in precedenza per riorganizzare la tabella master.
END_TABLE_REORGANIZATION	Garantisce che la tabella master della vista materializzata si trovi nello stato corretto e che sia valida alla fine di una riorganizzazione della tabella master.
EXPLAIN_MVIEW	Spiega ciò che è possibile con una vista materializzata esistente o proposta (è possibile l'aggiornamento rapido, è disponibile la riscrittura della query?).
EXPLAIN_REWRITE	Spiega perché la riscrittura di una query è fallita o quale vista materializzata verrà usata in caso di riscrittura.
I_AM_A_REFRESH	Viene restituito il valore dello stato del package I_AM_A_REFRESH chiamato durante la replicazione.
PMARKER	Usato per il Partition Change Tracking, restituisce un indicatore di partizione da un RowID.
PURGE_DIRECT_LOAD_LOG	Usato con il data warehousing, questo programma elimina le righe dal log di caricamento diretto una volta che non sono più richieste da una vista materializzata.
PURGE_LOG	Elimina righe dal log delle viste materializzate.
PURGE_MVIEW_FROM_LOG	Elimina righe dal log delle viste materializzate.
REFRESH	Aggiorna una o più viste materializzate che non sono membri dello stesso gruppo di aggiornamento.
REFRESH_ALL_MVIEW	Aggiorna tutte le viste materializzate che non riflettono i cambiamenti alla loro tabella master o alla vista materializzata master.
REFRESH_DEPENDENT	Aggiorna tutte le viste materializzate che non dipendono da una tabella master o da una vista materializzata master specifica. L'elenco può contenere una o più tabelle master o viste materializzate master.
REGISTER_MVIEW	Abilita l'amministrazione di una singola vista materializzata.
UNREGISTER_MVIEW	Usato per deregistrare una vista materializzata presso un sito master o il sito di una vista materializzata master.

```

| { cache | nocache }
]...
]
[clausola_parallel] [clausole_partizionamento]
[with
 { object id | primary key | rowid | sequence | ( colonna [, colonna]... ) }
 [, { object id | primary key | rowid | sequence | ( colonna [, colonna]... ) } ]...
]
[ { including | excluding } new values];

```

Il comando `create materialized view log` viene eseguito nel database della tabella master, in genere dal proprietario della tabella master. I log delle viste materializzate non dovrebbero essere creati per tabelle coinvolte solo in viste materializzate complesse (poiché non verrebbero utilizzate). Per il log delle viste materializzate non viene specificato alcun nome.

**Tabella 23.2** Sottoprogrammi di DBMS\_OLAP.

SOTTOPROGRAMMA	DESCRIZIONE
ADD_FILTER_ITEM	Filtra il contenuto usato durante il processo di raccomandazione.
CREATE_ID	Crea un ID interno utilizzato da una nuova collection di carico di lavoro, da un nuovo filtro o da una nuova esecuzione di Advisor.
ESTIMATE_MVIEW_SIZE	Stima la dimensione della vista materializzata in byte e righe che è possibile creare.
EVALUATE_MVIEW_STRATEGY	Misura l'utilizzo di ciascuna vista materializzata esistente.
GENERATE_MVIEW_REPORT	Genera un report HTML sull'esecuzione di Advisor data.
GENERATE_MVIEW_SCRIPT	Genera un semplice script contenente i comandi SQL per implementare le raccomandazioni fatte nel report di Summary Advisor.
LOAD_WORKLOAD_CACHE	Ottiene un carico di lavoro della cache SQL.
LOAD_WORKLOAD_TRACE	Carica un carico di lavoro raccolto da Oracle Trace.
LOAD_WORKLOAD_USER	Carica un carico di lavoro definito dall'utente.
PURGE_FILTER	Elimina un filtro specifico o tutti i filtri.
PURGE_RESULTS	Rimuove tutti i risultati o quelli di un'esecuzione specifica.
PURGE_WORKLOAD	Elimina una collection specifica o tutti i carichi di lavoro.
RECOMMEND_MVIEW_STRATEGY	Genera un set di raccomandazioni su quali viste materializzate dovrebbero essere create, mantenute o eliminate.
SET_CANCELLED	Interrompe l'Advisor. Se l'Advisor impiega troppo tempo per restituire i risultati, lo interrompe.
VALIDATE_DIMENSION	Verifica la correttezza delle relazioni specificate in una dimensione.
VALIDATE_WORKLOAD_CACHE	Convalida il carico di lavoro di SQL Cache prima di effettuare operazioni di caricamento.
VALIDATE_WORKLOAD_TRACE	Convalida il carico di lavoro di Oracle Trace prima di effettuare operazioni di caricamento.
VALIDATE_WORKLOAD_USER	Convalida il carico di lavoro fornito dall'utente prima di effettuare operazioni di caricamento.

Un log delle viste materializzate per la tabella BIBLIOTECA può essere creato con il comando riportato di seguito, eseguito dall'interno dell'account proprietario della tabella.

```
create materialized view log on BIBLIOTECA
  tablespace USERS
  storage (initial 40K next 40K pctincrease 0)
  with primary key;
```

Il comando di questo esempio crea un log delle viste materializzate in una tablespace di nome USERS. Dato che i log delle viste materializzate possono crescere con il tempo in modo imprevedibile, nei database di produzione, sarebbe opportuno memorizzare gli oggetti loro associati in tablespace dedicate ai log delle viste materializzate. La vista materializzata BIBLIOTECA\_LOCALE è stata creata con la clausola with primary key, quindi anche il log

delle viste materializzate per la tabella BIBLIOTECA dovrebbe essere creato con la medesima clausola.

## Privilegi di sistema richiesti

Per creare il log delle viste materializzate, sono necessari i privilegi di sistema CREATE TABLE e CREATE TRIGGER. Se si crea il log delle viste materializzate dall'account di un utente che non possiede la tabella master, è necessario disporre dei privilegi di sistema CREATE ANY TABLE, COMMENT ANY TABLE e CREATE ANY TRIGGER, nonché del privilegio SELECT sulla tabella master della vista materializzata.

## Oggetti locali e remoti

Quando si crea un log delle viste materializzate, nello schema della tabella master vengono creati due oggetti. Dal punto di vista del proprietario della vista materializzata, il comando `create materialized view log` viene eseguito nel database remoto e gli oggetti creati da questo comando si trovano tutti nel database remoto. Il log delle viste materializzate crea una tabella e un trigger.

- Viene creata una tabella per memorizzare gli identificatori delle righe modificate della tabella master, insieme a una colonna separata dell'indicatore orario che registra l'ora in cui sono state replicate per l'ultima volta le righe modificate.
- Viene creato un trigger interno per inserire i valori chiave, siano essi valori ROWID o valori di chiave primaria, e indicatori orari di aggiornamento delle righe aggiornate, inserite o eliminate nella tabella log delle viste materializzate.

Le colonne mostrate nel comando `create materialized view log` sono correlate all'uso di subquery all'interno di query eseguite su viste materializzate. Si può usare una subquery all'interno della query di una vista materializzata solo se la subquery mantiene completamente i valori chiave dei dati replicati. Le subquery possono essere utilizzate solo con viste materializzate basate su valori di chiave primaria.

## 23.8 Modifica dei log e delle viste materializzate

Per le viste materializzate già esistenti è possibile modificare i parametri di memorizzazione, le opzioni di aggiornamento e gli intervalli di aggiornamento. Se non si conoscono con esattezza le impostazioni correnti di una vista materializzata, si controlli la vista del dizionario dati `USER_MVIEWS`.

La sintassi del comando `alter materialized view` è mostrata nel Capitolo 42. Il comando del listato seguente modifica l'opzione di aggiornamento adottata dalla vista materializzata `BIBLIOTECA_LOCALE`:

```
alter materialized view BIBLIOTECA_LOCALE
refresh complete;
```

Tutti gli aggiornamenti futuri di `BIBLIOTECA_LOCALE` riguarderanno l'intera tabella base locale.

Per modificare una vista materializzata, è necessario esserne il proprietario, oppure disporre del privilegio di sistema `ALTER ANY MATERIALIZED VIEW`.

I parametri di memorizzazione per i log delle viste materializzate possono essere modificati tramite il comando `alter materialized view log`. La sintassi di questo comando è riportata nel Capitolo 42.

La modifica ai parametri di memorizzazione per un log delle viste materializzate comporterà la modifica dei parametri di memorizzazione sulla tabella del log delle viste materializzate. Per esempio, il comando seguente modificherà il parametro `next` nella clausola `storage` del log delle viste materializzate (assumendo che la tablespace sia gestita a livello di dizionario):

```
alter materialized view log on BIBLIOTECA  
storage (next 100K);
```

Per modificare un log delle viste materializzate, occorre possedere la tabella, disporre del privilegio `ALTER` per la tabella o avere il privilegio di sistema `ALTER ANY TABLE`.

## 23.9 Eliminazione di viste materializzate e log

Per eliminare una vista materializzata, è necessario disporre dei privilegi di sistema richiesti per eliminare sia la vista materializzata sia tutti gli oggetti ad essa correlati. Si deve disporre del privilegio `DROP MATERIALIZED VIEW` se l'oggetto si trova nel proprio schema, oppure del privilegio di sistema `DROP ANY MATERIALIZED VIEW` se la vista materializzata non si trova nel proprio schema.

Il comando seguente elimina la vista materializzata `CONTA_CATEGORIA_LOCALE` creata in precedenza in questo capitolo:

```
drop materialized view CONTA_CATEGORIA_LOCALE;
```

I log delle viste materializzate possono essere eliminati tramite il comando `drop materialized view log`. Una volta eliminato il log delle viste materializzate da una tabella master, non si potranno più eseguire aggiornamenti rapidi per le viste materializzate semplici basate su questa tabella. Sarebbe opportuno eliminare un log delle viste materializzate quando nessuna vista materializzata semplice si basa sulla tabella master. Il comando seguente elimina il log delle viste materializzate che era stato creato precedentemente sulla tabella `BIBLIOTECA`:

```
drop materialized view log on BIBLIOTECA;
```

Per eliminare un log delle viste materializzate, è necessario essere in grado di eliminare sia il log delle viste materializzate sia gli oggetti ad esso correlati. Se si è proprietari del log delle viste materializzate, si deve disporre dei privilegi di sistema `DROP TABLE` e `DROP TRIGGER`. Se invece non si è proprietari del log delle viste materializzate, è necessario disporre dei privilegi di sistema `DROP ANY TABLE` e `DROP ANY TRIGGER` per eseguire questo comando.

## Capitolo 24

# Uso di Oracle Text per le ricerche di testo

- 24.1 **Aggiunta di testo al database**
- 24.2 **Query di testo e indici di testo**
- 24.3 **Gruppi di indici**

**C**on l'aumento della quantità di testo nel database, cresce anche la complessità delle query di testo eseguite sul database. Invece di eseguire semplici corrispondenze di stringa, ora sono necessarie nuove caratteristiche di ricerca di testo, come l'attribuzione di priorità durante ricerche di più termini o l'ordinamento in graduatoria dei risultati di una ricerca di testo.

Per eseguire ricerche di testo è possibile usare Oracle Text. Nelle versioni precedenti, questa funzionalità era nota come ConText Cartridge e interMedia Text. Se si ha già una certa familiarità con ConText, la configurazione di Oracle Text potrebbe apparire più semplice e meglio integrata con il kernel di Oracle. A partire da Oracle9i sono state aggiunte nuove caratteristiche, come i gruppi di indici, per migliorare ulteriormente la capacità di ricerca di testo.

È possibile utilizzare Oracle Text per effettuare ricerche con caratteri jolly, corrispondenze non esatte, ordinamenti per rilevanza, ricerche di prossimità, ponderazione dei termini ed espansione delle parole. In questo capitolo si imparerà a configurare e utilizzare Oracle Text.

## 24.1 Aggiunta di testo al database

Il testo può essere aggiunto al database memorizzandolo fisicamente in una tabella o memorizzando nel database dei puntatori a file esterni. Ciò significa che, per quanto riguarda i libri nella biblioteca, esiste la possibilità di memorizzare le recensioni sia nel database sia in file esterni. Se si memorizzano le recensioni in file esterni, allora i nomi di file andranno memorizzati nel database.

Per memorizzare le recensioni nel database, si creino delle tabelle RECENSIONI\_LIBRI. In questo capitolo, verranno presentati esempi di due tipi di indici: CONTEXT e CTXCAT. A supporto di questi esempi, verranno create due tabelle separate: CONTEXT\_RECENSIONI\_LIBRI e CTXCAT\_RECENSIONI\_LIBRI, caricate entrambe con gli stessi dati.

```
create table CONTEXT_RECENSIONI_LIBRI
(Titolo      VARCHAR2(100) primary key,
Recensore    VARCHAR2(25),
Data_Recensione DATE,
Testo_Recensione  VARCHAR2(4000));
```

```
insert into CONTEXT_RECENSIONI_LIBRI values
('MY LEDGER', 'EMILY TALBOT', '01-MAY-02',
```

'Un'analisi molto interessante sulle transazioni e le finanze di G. B. Talbot e Dora Talbot

quando intorno al 1900 gestivano una proprietà nel New Hampshire. Le storie passano in rassegna le spese per i medicinali, le visite mediche e le tombe, per i lavoratori durante i raccolti e per i regali di Natale. Un gran bel libro. '');

```
create table CTXCAT_RECENSIONI_LIBRI
(Titolo      VARCHAR2(100) primary key,
Recensore    VARCHAR2(25),
Data_Recensione DATE,
Testo_Recensione VARCHAR2(4000));
```

```
insert into CTXCAT_RECENSIONI_LIBRI values
('MY LEDGER', 'EMILY TALBOT', '01-MAY-02',
'Un'analisi molto interessante sulle transazioni e le finanze di G. B. Talbot e Dora Talbot
quando intorno al 1900 gestivano una proprietà nel New Hampshire.
Le storie passano in rassegna le spese, per i medicinali, le visite mediche e le tombe, per
i lavoratori durante i raccolti e per i regali di Natale. Un gran bel libro.');
```

Alla colonna Testo\_Recensione delle tabelle RECENSIONI\_LIBRI viene assegnato il tipo di dati VARCHAR2(4000). Per valori più lunghi, si prenda in considerazione l'uso di tipi di dati CLOB. Per ulteriori informazioni sui tipi di dati CLOB, si rimanda al Capitolo 32. È possibile selezionare il testo della recensione dal database:

```
set linesize 70
select Testo_Recensione
  from CONTEXT_RECENSIONI_LIBRI
 where Titolo = 'MY LEDGER';
TESTO_RECENSIONE
```

---

Un'analisi molto interessante sulle transazioni e le finanze di G. B. Talbot e Dora Talbot quando intorno al 1900 gestivano una proprietà nel New Hampshire.

Le storie passano in rassegna le spese, per i medicinali, le visite mediche e le tombe, per i lavoratori durante i raccolti e per i regali di Natale. Un gran bel libro.

## 24.2 Query di testo e indici di testo

L'esecuzione di una query di testo è diversa dall'esecuzione di una query sui dati, perché le parole hanno sfumature di significato, relazioni con altre parole e contrari. Si possono cercare parole vicine le une alle altre, oppure parole correlate ad altre. Query simili sarebbero estremamente difficili se gli operatori relazionali standard fossero l'unico strumento disponibile. Con l'estensione di SQL agli indici di testo, Oracle Text consente di formulare domande molto complesse inerenti al testo.

Per usare Oracle Text, è necessario creare un *indice testuale* sulla colonna in cui il testo è memorizzato. "Indice testuale" è un termine leggermente fuorviante, in quanto si tratta di una collezione di tabelle e indici che memorizzano informazioni sul testo a sua volta memorizzato nella colonna.

In Oracle9i esistono molti tipi diversi di indici testuali. Il primo, CONTEXT, è supportato sia in Oracle8i sia in Oracle9i. A partire da Oracle9i è anche possibile usare l'indice testuale CTXCAT per migliorare ulteriormente la gestione degli indici testuali e le capacità di query. In questo capitolo verranno proposti esempi per entrambi gli indici. Si può utilizzare un terzo tipo di indice, CTXRULE, per costruire un'applicazione per la classificazione dei documenti basata

sul contenuto. Per i dettagli sull'uso degli indici CTXRULE si consulti la *Oracle Text Application Developer's Guide*.

**NOTA** *Prima di creare un indice testuale su una tabella, è necessario creare una chiave primaria per la tabella stessa (se non ne esiste già una).*

Tramite una versione speciale del comando `create index`, è possibile creare un indice testuale. Per un indice CONTEXT occorre specificare il tipo di indice Ctxsys.Context, come mostrato nel listato seguente:

```
create index Review_Context_Index on CONTEXT_RECENSIONI_LIBRI(Testo_Recensioni)
indextype is ctxsys.context;
```

Una volta creato l'indice testuale, Oracle genera diversi indici e tabelle nello schema dell'utente per supportare le query di testo. È possibile ricostruire l'indice testuale con il comando `alter index`, come si farebbe con un qualunque altro indice.

A partire da Oracle9i è possibile sostituire il tipo di indice CONTEXT con il tipo di indice CTXCAT:

```
create index Review_CtxCat_Index on CTXCAT_RECENSIONI_LIBRI(Testo_Recensione)
indextype is ctxsys.ctxcat;
```

Il tipo di indice CTXCAT supporta la sincronizzazione transazionale dei dati tra la tabella base (CTXCAT\_RECENSIONI\_LIBRI) e il suo indice testuale. Con gli indici CONTEXT è necessario comunicare manualmente a Oracle di aggiornare i valori nell'indice testuale dopo le modifiche dei dati apportate alla tabella base. I tipi di indici CTXCAT non creano valori di "punteggio" durante le query di testo (come invece fanno gli indici CONTEXT), ma la sintassi della query è davvero molto simile per entrambi i tipi di indici. I prossimi paragrafi illustrano i vari tipi di query di testo che è possibile eseguire con Oracle Text.

## Query di testo

Dopo la creazione di un indice testuale sulla colonna Testo\_Recensione della tabella CONTEXT\_RECENSIONI\_LIBRI, le capacità di ricerca del testo aumentano drasticamente. Ora, è possibile cercare qualsiasi recensione di libro che contenga la parola 'proprietà':

```
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, 'proprietà')>0;
```

La funzione CONTAINS accetta due parametri, il nome della colonna e la stringa di ricerca, e controlla l'indice testuale per la colonna Testo\_Recensione. Se nell'indice testuale della colonna Testo\_Recensione viene individuata la parola 'proprietà', il database restituirà un *punteggio* maggiore di zero e il valore di Titolo corrispondente. Il punteggio rappresenta una valutazione della corrispondenza del record restituito con i criteri specificati nella funzione CONTAINS.

Se si crea un indice CTXCAT, occorre utilizzare la funzione CATSEARCH al posto di CONTAINS. CATSEARCH accetta tre parametri: il nome della colonna, la stringa di ricerca e il nome del gruppo di indici. I gruppi di indici sono descritti nel paragrafo "Gruppi di indici" più avanti nel capitolo. Per questo esempio non viene utilizzato nessun gruppo di indici, quindi il parametro corrispondente è impostato a NULL:

```
select Titolo
  from CTXCAT_RECENSIONI_LIBRI
 where CATSEARCH(Testo_Recensione, 'proprietà', NULL)>0;
```

CATSEARCH non calcola un punteggio ma usa la sintassi 0, semplificando la migrazione dagli indici CONTEXT verso quelli CTXCAT.

### **Funzionamento di una query di testo**

Quando in una query si usa una funzione come CONTAINS o CATSEARCH, la porzione testuale della query viene elaborata da Oracle Text. Il resto della query viene elaborato esattamente come una query normale all'interno del database. I risultati dell'elaborazione della query di testo e dell'elaborazione della query normale vengono raccolti, in modo da restituire un unico gruppo di record all'utente.

### **Espressioni di query di testo disponibili**

Oracle Text sarebbe piuttosto limitato se consentisse esclusivamente la ricerca di parole che coincidono con esattezza. In realtà, per la ricerca del testo Oracle Text mette a disposizione un'ampia varietà di soluzioni che è possibile utilizzare per personalizzare le query. La maggior parte delle soluzioni per la ricerca del testo è attivata tramite le funzioni CONTAINS e CATSEARCH, che possono apparire solo nella clausola where di un'istruzione select, mai nelle clausole where di insert, update o delete.

Gli operatori all'interno della funzione CONTAINS consentono di eseguire le ricerche di testo seguenti.

- Corrispondenze esatte di una parola o frase.
- Corrispondenze esatte di più parole, utilizzando la logica booleana per combinare le ricerche.
- Ricerche basate sulla somiglianza reciproca delle parole nel testo.
- Ricerche di parole che condividono la stessa etimologia.
- Corrispondenze non esatte delle parole.
- Ricerche di parole che abbiano lo stesso suono di altre.

CATSEARCH supporta le funzioni per la ricerca di corrispondenze esatte così come la creazione di "gruppi di indici", descritti più avanti in questo capitolo. Nei paragrafi seguenti, sono riportati esempi di questi tipi di ricerche testuali, insieme a informazioni sugli operatori che si possono usare per personalizzare le ricerche di testo.

### **Ricerca di una corrispondenza esatta di una parola**

La query seguente, eseguita sulle tabelle RECENSIONI\_LIBRI, restituisce il titolo di tutte le recensioni che includono la parola "proprietà":

```
REM Metodo CONTAINS per gli indici CONTEXT:
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, 'proprietà')>0;
```

```
REM Metodo CATSEARCH per gli indici CTXCAT:
select Titolo
  from CTXCAT_RECENSIONI_LIBRI
 where CATSEARCH(Testo_Recensione, 'proprietà', NULL)>0;
```

All'interno delle chiamate a funzione, il simbolo ‘>’ viene detto *operatore di soglia*. La precedente ricerca di testo può essere tradotta nel modo seguente:

```
seleziona tutti i valori della colonna Titolo
dalla tabella CONTEXT_RECENSIONI_LIBRI
in cui il punteggio della ricerca di testo sulla colonna Testo_Recensione
per una corrispondenza esatta con la parola 'proprietà'
superia un valore soglia pari a 0.
```

L'analisi di soglia confronta il punteggio, ossia il punteggio interno calcolato da Oracle al momento dell'esecuzione della ricerca di testo, con il valore di soglia specificato. I valori di punteggio per ricerche individuali vanno da 0 a 10 per ciascuna occorrenza della stringa di ricerca all'interno del testo esaminato. Per gli indici CONTEXT, è possibile visualizzare il punteggio come parte della query.

Per visualizzare il punteggio di una ricerca di testo, occorre utilizzare la funzione SCORE, che ha un solo parametro, un'etichetta che si assegna al punteggio all'interno della ricerca di testo:

```
column Titolo format a30

select Titolo, PUNTEGGIO(10)
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, proprietà, 10)>0;
```

TITOLO	PUNTEGGIO(10)
MY LEDGER	3

In questo listato, i parametri della funzione CONTAINS vengono modificati per contenere un'etichetta ('10') per l'operazione di ricerca di testo eseguita. La funzione SCORE visualizza il punteggio della ricerca di testo associata a tale etichetta.

Per gli indici CONTEXT, è possibile utilizzare la funzione SCORE nell'elenco select (come nella query precedente), in una clausola group by o in una clausola order by.

## Ricerca di una corrispondenza esatta di più parole

Cosa si deve fare se si intende esaminare il testo per cercare più parole? Si può applicare la logica booleana (AND e OR) per combinare i risultati di più ricerche di testo in un'unica query. Inoltre, si possono cercare più termini all'interno delle stesse chiamate a funzione e fare in modo che Oracle risolva i risultati della ricerca.

Per esempio, se si desidera cercare le recensioni che contengono le parole “proprietà” e “raccolti” nel testo della recensione, si potrebbe inserire la query seguente:

```
REM Metodo CONTAINS per gli indici CONTEXT:
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, 'proprietà AND raccolti')>0;
```

```
REM Metodo CATSEARCH per gli indici CTXCAT:
select Titolo
  from CTXCAT_RECENSIONI_LIBRI
 where CATSEARCH(Testo_Recensione, 'proprietà AND raccolti', NULL)>0;
```

**NOTA** Questa query non cerca la frase “proprietà e raccolti”, bensì le due singole parole all’interno di tutto il testo esaminato. Nel prossimo paragrafo si analizzerà la sintassi per le ricerche di frasi.

Nella query dell’indice CONTEXT, al posto di AND si sarebbe potuto usare il simbolo della e commerciale (&). Prima di usare questo metodo in SQLPLUS, si imposti il comando set define off in modo che il carattere & non compaia come parte del nome di una variabile:

```
set define off
```

```
REM Metodo CONTAINS per gli indici CONTEXT:  
select Titolo  
  from CONTEXT_RECENSIONI_LIBRI  
 where CONTAINS(Testo_Recensione, 'proprietà & raccolti')>0;
```

Per l’indice CTXCAT, si può tralasciare completamente la parola AND:

```
REM Metodo CATSEARCH per gli indici CTXCAT:  
select Titolo  
  from CTXCAT_RECENSIONI_LIBRI  
 where CATSEARCH(Testo_Recensione, 'proprietà raccolti', NULL)>0;
```

L’uso del carattere & o della parola AND denota un’operazione AND, pertanto la funzione CONTAINS restituisce una riga solo se il testo della recensione include *entrambe* le parole “proprietà” e “raccolti”. Ciascuna ricerca dovrà superare i criteri di soglia definiti per i punteggi di ricerca. Per cercare più di due termini, sarà sufficiente aggiungerli alla clausola CONTAINS o CATSEARCH, come mostrato nel listato seguente:

```
REM Metodo CONTAINS per gli indici CONTEXT:  
select Titolo  
  from CONTEXT_RECENSIONI_LIBRI  
 where CONTAINS(Testo_Recensione, 'proprietà AND raccolti AND lavoratori')>0;  
REM Metodo CATSEARCH per gli indici CTXCAT:  
select Titolo  
  from CTXCAT_RECENSIONI_LIBRI  
 where CATSEARCH(Testo_Recensione, 'proprietà raccolti lavoratori', NULL)>0;
```

La query di questo listato restituisce una riga solo se i punteggi di ricerca per “proprietà”, “raccolti” e “lavoratori” sono tutti maggiori di 0.

Oltre ad AND, è possibile utilizzare l’operatore OR, nel qual caso viene restituito un record se una delle condizioni di ricerca soddisfa la soglia definita. In Oracle Text, il simbolo per l’operatore OR è una barra verticale (|), quindi le due query seguenti sono elaborate allo stesso modo.

```
REM Metodo CONTAINS per gli indici CONTEXT:  
select Title  
  from BOOK_REVIEW_CONTEXT  
 where CONTAINS(Review_Text, proprietà OR raccolti)>0;  
  
select Titolo  
  from CONTEXT_RECENSIONI_LIBRI  
 where CONTAINS(Testo_Recensione, 'proprietà | raccolti')>0;
```

Quando si eseguono queste query, viene restituito un record se una delle due ricerche separate (“proprietà” e “raccolti”) restituisce un punteggio maggiore di 0. La funzione CATSEARCH non supporta l’uso del termine or; ma solo il simbolo |:

```
REM Metodo CATSEARCH per gli indici CTXCAT:
```

```
select Titolo
  from CTXCAT_RECENSIONI_LIBRI
 where CATSEARCH(Testo_Recensione, 'proprietà | raccolti', NULL)>0;
```

L’operatore ACCUM (“accumula”) mette a disposizione un altro metodo per la combinazione delle ricerche. Questo operatore somma i punteggi delle singole ricerche e confronta il punteggio accumulato con il valore di soglia. Il simbolo per l’operatore ACCUM è una virgola (,). Le due query seguenti sono pertanto equivalenti.

```
REM Metodo CONTAINS per gli indici CONTEXT:
```

```
select Title
  from BOOK_REVIEW_CONTEXT
 where CONTAINS(Review_Text, proprietà ACCUM raccolti')>0;
```

```
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, 'proprietà , raccolti')>0;
```

La sintassi di ACCUM è supportata nelle chiamate alla funzione CATSEARCH, ma non dovrebbe essere utilizzata, poiché CATSEARCH non calcola un punteggio da confrontare poi con il valore di soglia.

Oracle Text può anche essere usato per sottrarre i punteggi da più ricerche prima di confrontare il risultato con il punteggio di soglia. L’operatore MINUS in CONTAINS sottrae il punteggio della ricerca del secondo termine dal punteggio della ricerca del primo. Le query del listato seguente determinano il punteggio di ricerca del termine “proprietà”, sottraggono da esso il punteggio di ricerca del termine “casa” e infine confrontano la differenza con il punteggio di soglia.

```
REM Metodo CONTAINS per gli indici CONTEXT:
```

```
select Title
  from BOOK_REVIEW_CONTEXT
 where CONTAINS(Review_Text, proprietà MINUS casa')>0;
```

```
select Titolo
  from CONTEXT:RECENSIONI:LIBRI
 where CONTAINS(Testo_Recensione, 'proprietà - casa')>0;
```

Al posto dell’operatore MINUS è possibile inserire il simbolo ‘-’, come mostrato nel listato precedente.

Per gli indici CONTEXT, l’operatore – riduce il punteggio quando si confronta il punteggio di ricerca totale con il valore di soglia, ma non smette di considerare la riga in questione. Per eliminare le righe in base ai termini di ricerca negli indici CONTEXT, occorre utilizzare il simbolo “~” come operatore “non”.

Per gli indici CTXCAT, il simbolo “-” ha un significato diverso di quello che assume per gli indici CONTEXT. Nel caso degli indici CTXCAT, “-” comunica a Oracle Text di non restituire la riga se il termine di ricerca si trova dopo “-” (come “~” negli indici CONTEXT). Se si trova

il secondo termine, CATSEARCH non restituisce la riga. Per le query CATSEARCH, è possibile sostituire “–” con la parola “not”.

Per rendere più chiara la logica dei criteri di ricerca è possibile utilizzare le parentesi. Se la ricerca contiene sia operatori AND sia operatori OR, è opportuno inserire delle parentesi per rendere più chiaro il modo in cui le righe vengono elaborate. Per esempio, la query seguente restituisce una riga se il testo esaminato contiene la parola ‘casa’ o contiene entrambe le parole ‘lavoratori’ e ‘raccolti’.

```
REM Metodo CONTAINS per gli indici CONTEXT:
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, 'casa OR (lavoratori AND raccolti')>0;
```

CATSEARCH non richiede l’inserimento della parola AND tra i termini “lavoratori” e “raccolti”:

```
REM Metodo CATSEARCH per gli indici CTXCAT:
select Titolo
  from CTXCAT_RECENSIONI_LIBRI
 where CATSEARCH(Testo_Recensione, 'casa | (lavoratori raccolti'), NULL)>0;
```

Modificando la posizione delle parentesi si modifica anche la logica della ricerca di testo. La query seguente restituisce una riga se il testo esaminato contiene la parola ‘casa’ o la parola ‘lavoratori’ e contiene anche la parola ‘raccolti’:

```
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, '(casa OR lavoratori) AND raccolti')>0;
```

Quando si valutano i punteggi di più ricerche negli indici CONTEXT, si può comunicare a Oracle Text di valutare i punteggi di alcune ricerche come più importanti di altri. Per esempio, se si desidera che il punteggio di ricerca del termine ‘raccolti’ valga il doppio quando confrontato con il punteggio di soglia, si può utilizzare il simbolo asterisco (\*) per indicare il fattore per cui il punteggio di ricerca dev’essere moltiplicato.

La query seguente raddoppia il punteggio di ricerca per ‘raccolti’ nella valutazione all’interno di una condizione OR.

```
select Titolo, PUNTEGGIO(10)
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, 'raccolti*2 OR proprietà*1',10)>5;
```

Con gli indici CONTEXT, è possibile valutare i punteggi di ricerca in modo che indichino la rilevanza dei termini nella ricerca. Se un termine è il più importante nella ricerca, è possibile assegnare a tale termine il “peso” maggiore. Grazie agli operatori AND, OR, ACCUM e MINUS è possibile ricercare qualsiasi combinazione di corrispondenze dei termini. Nel paragrafo seguente è descritta la ricerca di frasi.

## Ricerca di una corrispondenza esatta di una frase

Quando si cerca un’esatta corrispondenza di una frase, è necessario specificare l’intera frase come parte della stringa di ricerca. Se la frase include delle parole riservate (come ‘and’, ‘or’ oppure ‘minus’), sarà necessario utilizzare dei caratteri di escape, descritti in questo paragrafo, in modo che la ricerca venga effettuata correttamente.

La query seguente cerca qualsiasi titolo la cui voce di recensione contenga la frase “visite mediche”.

```
REM Metodo CONTAINS per gli indici CONTEXT:
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, 'visite mediche')>0;
REM Metodo CATSEARCH per gli indici CTXCAT:
select Titolo
  from CTXCAT_RECENSIONI_LIBRI
 where CATSEARCH(Testo_Recensione, 'visite mediche', NULL)>0;
```

Se la frase cercata contiene una parola riservata in Oracle Text come, “and” o “or”, si dovranno inserire delle parentesi graffe ({} ) per racchiudere il testo. La query seguente presuppone che la recensione sia in inglese e cerca la frase ‘transactions and finances’. La parola ‘and’ è racchiusa tra parentesi graffe.

```
REM Metodo CONTAINS per gli indici CONTEXT:
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, 'transactions {and} finances')>0;
REM Metodo CATSEARCH per gli indici CTXCAT:
select Titolo
  from CTXCAT_RECENSIONI_LIBRI
 where CATSEARCH(Testo_Recensione, transactions {and} finances', NULL)>0;
```

La query ‘transactions {and} finances’ è diversa dalla query ‘transactions and finances’. La query ‘transactions {and} finances’ restituisce un record solo se nel testo esaminato esiste la frase ‘transactions and finances’. La query ‘transactions and finances’ restituisce un record se il punteggio di ricerca del *termine* “transactions” e quello del *termine* “finances” superano entrambi il punteggio di soglia (o con un indice CTXCAT, se si trovano entrambi).

È possibile racchiudere l’intera frase tra parentesi graffe, nel qual caso qualsiasi parola riservata all’interno della frase stessa verrà trattata come parte del criterio di ricerca, come mostrato nell’esempio seguente:

```
REM Metodo CONTAINS per gli indici CONTEXT:
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, '{transactions and finances}')>0;
```

## Ricerche di parole vicine le une alle altre

La capacità *ricerca di prossimità* può essere utilizzata per eseguire una ricerca di testo basata sulla vicinanza dei termini cercati nel documento esaminato. Una ricerca di prossimità restituisce un punteggio elevato per parole adiacenti, e un punteggio basso per parole molto lontane. Se le parole sono adiacenti, la ricerca di prossimità restituisce un punteggio pari a 100.

Per eseguire una ricerca di prossimità negli indici CONTEXT, si utilizzi la parola chiave NEAR, come nell’esempio seguente:

```
REM Metodo CONTAINS per gli indici CONTEXT:
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, 'lavoratori NEAR raccolti')>0;
```

L'operatore NEAR può essere sostituito con il suo simbolo equivalente, il punto e virgola (;). Di seguito è riportata la query modificata:

```
REM Metodo CONTAINS per gli indici CONTEXT:  
select Titolo  
  from CONTEXT_RECENSIONI_LIBRI  
 where CONTAINS(Testo_Recensione, 'lavoratori; raccolti')>0;
```

Nelle query degli indici CONTEXT è possibile specificare il numero massimo di parole tra i termini di ricerca. Per esempio, per le parole separate da 10 termini, si potrebbe applicare una stringa di ricerca come 'NEAR((lavoratori, raccolti),10)'.

I metodi di ricerca di frasi e parole descritti in questo capitolo possono essere utilizzati per cercare corrispondenze esatte di parole e frasi e per effettuare ricerche di prossimità di parole e frasi esatte. Finora tutte le ricerche hanno utilizzato come base corrispondenze esatte dei termini cercati. Nei prossimi quattro paragrafi, si imparerà a espandere i termini di ricerca tramite quattro metodi: caratteri jolly, radici delle parole, corrispondenze non esatte e ricerche SOUNDEX.

## **Uso dei caratteri jolly durante le ricerche**

Negli esempi precedenti del capitolo, le query selezionavano valori di testo che corrispondevano esattamente ai criteri specificati. Per esempio, i termini di ricerca includevano "lavoratori" e non "lavoratore". I caratteri jolly consentono di espandere l'elenco dei termini validi per la ricerca da utilizzarsi nella query.

Come nella normale elaborazione di caratteri jolly in stringhe di testo, sono disponibili due caratteri jolly.

CARATTERE	DESCRIZIONE
%	Simbolo di percentuale: carattere jolly che sta per più caratteri.
-	Trattino di sottolineatura: carattere jolly che sta per singolo carattere.

La query seguente cerca tutte le corrispondenze di testo per tutte le parole che iniziano con i caratteri 'lavor'.

```
REM Metodo CONTAINS per gli indici CONTEXT:  
select Titolo  
  from CONTEXT_RECENSIONI_LIBRI  
 where CONTAINS(Testo_Recensione, 'lavor%')>0;
```

La query seguente limita l'espansione della stringa di testo esattamente a tre caratteri. Al posto del simbolo di percentuale %, utilizzato nella query precedente, vengono inseriti tre trattini di sottolineatura (\_ \_ \_). Dato che il trattino di sottolineatura è un carattere jolly per il singolo carattere, durante la ricerca la stringa di testo non potrà espandersi oltre i tre caratteri. Per esempio, la ricerca di testo potrebbe restituire la parola 'lavoro', mentre la parola 'lavoratore' è troppo lunga per essere restituita.

```
REM Metodo CONTAINS per gli indici CONTEXT:  
select Titolo  
  from CONTEXT_RECENSIONI_LIBRI  
 where CONTAINS(Testo_Recensione, 'lavor___')>0;
```

I caratteri jolly devono essere utilizzati quando si è certi della presenza di alcuni caratteri all'interno della stringa di ricerca. Se non si è sicuri della stringa di ricerca, è possibile utilizzare uno dei metodi descritti nei paragrafi seguenti: radici delle parole, corrispondenze non esatte e le corrispondenze SOUNDEX.

## Ricerca di parole con la stessa radice

Al posto dei caratteri jolly, è possibile utilizzare le capacità di *espansione etimologica* per ampliare l'elenco delle stringhe di testo. Conoscendo la “radice” di una parola, Oracle amplierà l'elenco delle parole da cercare sino a includervi tutte le parole che hanno la stessa radice. Ecco alcune espansioni di esempio (questa funzionalità è stata pensata per la lingua inglese):

RADICE	ESEMPI DI ESPANSIONE
Play	plays played playing playful
Works	working work worked workman workhorse
Have	had has haven't hasn't
Story	stories

Dato che ‘works’ e ‘work’ hanno la stessa radice, una ricerca con espansione etimologica che utilizza la parola ‘works’ restituirà anche il testo contenente la parola ‘work’.

Per usare l'espansione etimologica all'interno di una query, occorre inserire il simbolo del dollaro (\$). All'interno della stringa di ricerca il simbolo ‘\$’ deve precedere immediatamente la parola da espandere, senza spazi tra il simbolo e la parola stessa.

Il listato seguente mostra il risultato di una query eseguita sulla tabella CONTEXT\_RECENSIONI\_LIBRI per tutte le recensioni che contengono una parola che condivide la radice della parola “manage”:

```
REM Metodo CONTAINS per gli indici CONTEXT:
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, '$manage')>0;
```

Quando si esegue questa query, Oracle espande la parola “\$manage” fino a includere tutte le parole che hanno la stessa radice, quindi esegue la ricerca. Se una recensione include una delle parole con una radice “manage”, il record verrà restituito all’utente.

L'espansione dei termini di ricerca tramite le radici delle parole semplifica il processo di query. Non è più necessario sapere in quale forma un verbo o un termine sia stato utilizzato al momento dell'inserimento del testo: tutte le forme serviranno come base per la ricerca. Non è necessario specificare stringhe di testo particolari, come avviene per le query relative a corrispondenze esatte o quando si utilizzano i caratteri jolly. È sufficiente specificare una parola, e Oracle Text determinerà dinamicamente tutte le parole che devono essere cercate, basandosi su quella specificata.

## Ricerca di corrispondenze non esatte

Una *corrispondenza non esatta* (*fuzzy match*) espande il termine di ricerca specificato sino a includere parole scritte in modo simile, ma che non hanno necessariamente la stessa radice. Le

corrispondenze non esatte sono le più utili quando il testo contiene errori di ortografia. Tali errori possono trovarsi sia nel testo esaminato, sia nella stringa di ricerca specificata dall'utente durante la query.

Per esempio, questa query non restituirà ‘MY LEDGER’ perché la sua recensione non contiene la parola “racconti”:

```
REM Metodo CONTAINS per gli indici CONTEXT:  
select Titolo  
  from CONTEXT_RECENSIONI_LIBRI  
 where CONTAINS(Testo_Recensione, 'racconti')>0;
```

Tuttavia, contiene la parola “raccolti”. Una corrispondenza non esatta restituirà le recensioni contenenti la parola ‘raccolti’ anche se ha una radice di parola differente da quella della parola utilizzata come termine di ricerca.

Per usare una corrispondenza non esatta, è necessario anteporre al termine di ricerca un punto interrogativo, senza spazi tra il punto di domanda e l'inizio del termine di ricerca. L'esempio seguente illustra l'uso della capacità di corrispondenza non esatta.

```
REM Metodo CONTAINS per gli indici CONTEXT:  
select Titolo  
  from CONTEXT_RECENSIONI_LIBRI  
 where CONTAINS(Testo_Recensione, '?racconti')>0;
```

## Ricerche di parole dal suono simile ad altre parole

Le ricerche con espansione etimologica ampliano la ricerca di un termine fino a comprendere più termini basati sulla radice della parola. Le corrispondenze non esatte espandono un termine di ricerca basandosi sulle parole a questo simili presenti nell'indice testuale. Un terzo tipo di espansione del termine di ricerca, la ricerca SOUNDEX, amplia la ricerca basandosi sul suono della parola (anche questa funzionalità è valida per la lingua inglese). Il metodo di espansione di SOUNDEX utilizza la stessa logica di corrispondenza testuale messa a disposizione dalla funzione SQL SOUNDEX.

Per usare l'opzione SOUNDEX, è necessario anteporre al termine di ricerca un punto esclamativo (!), senza spazi tra il termine di ricerca e il punto esclamativo. Durante la ricerca, Oracle valuta i valori SOUNDEX dei termini presenti nell'indice testuale e cerca tutte le parole che abbiano lo stesso valore.

Come mostrato nella query seguente, si possono cercare tutte le recensioni che includono la parola “great”, usando una tecnica di corrispondenza SOUNDEX:

```
REM Metodo CONTAINS per gli indici CONTEXT:  
select Titolo  
  from CONTEXT_RECENSIONI_LIBRI  
 where CONTAINS(Testo_Recensione, '!grate')>0;
```

Se una recensione contenesse la parola “great”, verrebbe restituita come risultato, in quanto le parole “grate” e “great” hanno lo stesso valore SOUNDEX.

Gli operatori possono anche essere *annidati*, consentendo l'esecuzione di espansioni etimologiche sui termini restituiti da una corrispondenza non esatta. Nell'esempio seguente, viene eseguita una corrispondenza non esatta della parola ‘stir’. I termini restituiti dalla corrispondenza non esatta vengono quindi ampliati utilizzando l'espansione etimologica.

```
REM Metodo CONTAINS per gli indici CONTEXT:  
select Titolo  
  from CONTEXT_RECENSIONI_LIBRI  
 where CONTAINS(Testo_Recensione, '$?stir')>0;
```

Le principali opzioni di ricerca per la funzione CONTAINS sono riassunte nella Tabella 24.1. Per un elenco di tutta la sintassi supportata (incluso l'uso del thesaurus, l'espansione dei sinonimi e il supporto XML), si consulti la guida *Oracle Text Reference*. Le opzioni di ricerca della funzione CATSEARCH sono elencate nella Tabella 24.2. Questa tabella non elenca le caratteri-

**Tabella 24.1** Operatori principali di CONTAINS.

OPERATORE	DESCRIZIONE
OR	Restituisce un record se uno dei termini di ricerca ha un punteggio superiore alla soglia.
	Come OR.
AND	Restituisce un record se entrambi i termini di ricerca hanno un punteggio superiore alla soglia.
&	Come AND.
ACCUM	Restituisce un record se la somma dei punteggi dei termini di ricerca supera la soglia.
,	Come ACCUM.
MINUS	Restituisce un record se il punteggio della prima ricerca meno quello della seconda supera la soglia.
-	Come MINUS.
*	Assegna pesi differenti ai punteggi delle ricerche.
NEAR	Il punteggio sarà basato sulla vicinanza dei termini di ricerca nel testo.
:	Come NEAR.
NOT	Esclude la riga se trova il termine dopo NOT.
~	Come NOT.
EQUIV	Tratta due termini (termine1 equiv termine2) come uguali nella determinazione del punteggio della ricerca.
=	Come EQUIV.
{}	Racchiude le parole riservate come AND se fanno parte del termine di ricerca.
%	Carattere jolly che indica un numero qualsiasi di caratteri.
_	Carattere jolly che indica un carattere singolo.
\$	Esegue l'espansione della radice del termine di ricerca prima di eseguire la ricerca stessa.
?	Segnala di cercare una corrispondenza non esatta prima della ricerca.
!	Esegue una ricerca SOUNDEX.
()	Specifica l'ordine in cui sono valutati i criteri di ricerca.

**Tabella 24.2** Opzioni di CATSEARCH.

OPERATORE	DESCRIZIONE
	Restituisce un record se trova uno dei termini di ricerca.
AND	Restituisce un record se trova entrambi i termini di ricerca. Questa è l'azione predefinita, quindi (a b) viene trattato come (a AND b).
-	Restituisce le righe che contengono il termine che precede l'operatore “-” e che non contengono il termine che lo segue.
NOT	Identico a -.
“ ”	Racchiude delle frasi.
()	Specifica l'ordine di valutazione dei criteri di ricerca.

stiche di CONTAINS disapprovate, come ACCUM, che non sono documentate come caratteristiche supportate per CATSEARCH.

## Uso dell'operatore ABOUT

In Oracle Text è possibile cercare le tematiche dei documenti. La ricerca tematica è integrata con la ricerca testo-termine. Per cercare parole che hanno a che fare con il tema del documento, invece che termini specifici all'interno del documento stesso, si può utilizzare l'operatore ABOUT. Per esempio:

```
REM Metodo CONTAINS per gli indici CONTEXT:
select Testo_Recensione

from CONTEXT_RECENSIONI_LIBRI
where CONTAINS(Testo_Recensione, 'ABOUT(medicina)')>0;

REM Metodo CATSEARCH per gli indici CTXCAT:
select Titolo
  from CTXCAT_RECENSIONI_LIBRI
 where CATSEARCH(Testo_Recensione, 'ABOUT(medicina)', NULL)>0;
```

## Sincronizzazione degli indici

Con gli indici CONTEXT è necessario gestire il contenuto degli indici testuali; gli indici testuali non vengono aggiornati mentre si aggiorna la tabella. Non appena un valore di Testo\_Recensione viene aggiornato, il suo indice testuale non è più sincronizzato con la tabella base. Per sincronizzare l'indice, è necessario eseguire la procedura SYNC\_INDEX del package CTX\_DDL, come mostrato nel listato seguente:

```
execute CTX_DDL.SYNC_INDEX('INDICE_RECENSIONE');
```

## 24.3 Gruppi di indici

Storicamente, i problemi con le query degli indici testuali si verificavano quando insieme alle ricerche di testo, come parte della clausola `where`, venivano utilizzati anche altri criteri. Per esempio, le clausole `where` sono state applicate su colonne non di testo prima o dopo il completamento delle ricerche di testo? E come è possibile ordinare in modo corretto i risultati? Per migliorare la capacità di query “mista”, Oracle9i presenta i gruppi di indici. Gli indici compresi in un gruppo di indici possono trovarsi su colonne relazionali strutturate oppure su colonne di testo.

Per creare un gruppo di indici, si utilizzi il package `CTX_DDL`, che serve per creare il gruppo di indici e aggiungere a quest’ultimo degli indici. Quando si crea un indice testuale, è possibile specificare il gruppo di indici al quale appartiene. Per eseguire questo esempio, per prima cosa si dovrebbe eliminare l’indice testuale `REVIEW_CTXCAT_INDEX` creato in precedenza in questo capitolo sulla tabella `CTXCAT_RECENSIONI_LIBRI`:

```
drop index REVIEW_CTXCAT_INDEX;
```

Per creare un gruppo di indici di nome `Recensioni`, è necessario utilizzare la procedura `CREATE_INDEX_SET`:

```
execute CTX_DDL.CREATE_INDEX_SET('Recensioni');
```

Ora si possono aggiungere degli indici al gruppo di indici tramite la procedura `ADD_INDEX`. Per prima cosa, si aggiunga un indice non testuale standard:

```
execute CTX_DDL.ADD_INDEX('Recensioni', 'Recensore');
execute CTX_DDL.ADD_INDEX('Recensioni', 'Data_Recensione');
```

Ora, si crei un indice testuale `CTXCAT`. Si specifichi `Ctxsys.Ctxcat` come tipo di indice e si elenchi il gruppo di indici nella clausola `parameters`:

```
create index REVIEW_CTXCAT_INDEX
  on CTXCAT_RECENSIONI_LIBRI (Testo_Recensione)
  indextype is CTXSYS.CTXCAT
parameters ('gruppo indici Recensioni');
```

Ora, si possono ordinare i risultati in base al risultato della ricerca di gruppi di indici combinate.

```
select * from CTXCAT_RECENSIONI_LIBRI
  where CATSEARCH(Testo_Recensione, 'great',
    'Recensore' 'EMILY TALBOT'
    order by Data_Recensione desc')>0;
```

Per risolvere questa query, Oracle Text utilizzerà il gruppo di indici concedendo la possibilità di ordinare i risultati in modo corretto. Si osservi che intorno alla stringa ‘EMILY TALBOT’ ci sono delle virgolette, ma poiché si tratta della ricerca di una stringa di testo, quando si esegue la query queste virgolette verranno convertite in un gruppo di apici.

I gruppi di indici possono contenere fino a 99 indici sui tipi di dati NUMBER, DATE, CHAR e VARCHAR2. Nessuna colonna, in un indice che faccia parte di un gruppo di indici, può

superare i 30 byte (pertanto, in questo esempio la colonna Titolo non potrà essere indicizzata come parte di un gruppo di indici). Le colonne indicizzate non devono contenere valori NULL.

Per ulteriori dettagli sui gruppi di indici, la gestione degli indici testuali e lo sviluppo di applicazioni di testo, si consulti la *Oracle Text Application Developer's Guide* e la *Oracle Text Reference*, entrambe fornite come parte della documentazione standard di Oracle.

## Capitolo 25

# Uso di tabelle esterne

- 25.1 **Accesso ai dati esterni**
- 25.2 **Creazione di una tabella esterna**
- 25.3 **Limitazioni, vantaggi e usi possibili delle tabelle esterne**

**A** partire da Oracle9i è possibile utilizzare la funzionalità delle tabelle esterne, con le quali si può accedere ai file esterni come se questi fossero tabelle contenute nel database. Quando si crea una tabella esterna, la sua struttura viene definita insieme alla sua posizione in Oracle. Quando si esegue la query sulla tabella, Oracle legge la tabella esterna e restituisce i risultati come se i dati fossero stati memorizzati nel database. Dal momento però che i dati non sono nel database, non ci si dovrà preoccupare di caricarli nel database, un vantaggio che potrebbe essere davvero notevole per le data warehouse e per i database di grosse dimensioni.

Le tabelle esterne hanno dei limiti; infatti non è possibile inserire, aggiornare o cancellare le loro righe dall'interno di Oracle; inoltre non le si può neppure indicizzare. Dato che fanno parte delle applicazioni di database, le si dovrà considerare come parti integranti dei processi di backup e di recupero. Nonostante queste complicazioni, le tabelle esterne possono essere un'aggiunta davvero potente ai piani dell'architettura del database.

## 25.1 Accesso ai dati esterni

Per accedere ai file esterni dall'interno di Oracle, per prima cosa occorre utilizzare il comando `create directory` per definire l'oggetto di una directory che punta alla posizione del file esterno. Gli utenti che accederanno ai file esterni dovranno possedere il privilegio `READ` nella directory.

**NOTA** *Prima di iniziare, occorre accertarsi dell'esistenza della directory esterna e che l'utente che eseguirà il comando `create directory` abbia il privilegio di sistema `CREATE ANY DIRECTORY`.*

L'esempio seguente crea una directory di nome `LIBRO_DIR` e concede gli accessi `READ` e `WRITE` allo schema Pratica:

```
create directory LIBRO_DIR as 'e:\oracle\external';
grant read on directory LIBRO_DIR to pratica;
grant write on directory LIBRO_DIR to pratica;
```

Ora l'utente di Pratica può leggere i file nella directory `e:\oracle\external` come se fossero contenuti nel database. Dato che a Pratica è stato concesso anche il privilegio `WRITE` in quella directory, l'utente di Pratica potrà creare file di log, file di scarto e file non validi nella directory, come se stesse eseguendo l'utilità `SQL*Loader` (si consulti il Capitolo 21).

Il listato seguente genera due file per dei dati di esempio, uno tratto da BIBLIOTECA e uno da BIBLIOTECA\_AUTORE. Si osservi che il comando spool non può utilizzare il nome della directory creata con create directory; si dovrà specificare il nome completo della directory del sistema operativo.

```
connect pratica/pratica

set pagesize 0 newpage 0 feedback off
select Titolo||'~'||Editore||'~'||NomeCategoria||'~'||Classificazione||'~'
  from BIBLIOTECA
 order by Titolo

spool e:\oracle\external\bookshelf_dump.lst
/
spool off

select Titolo||'~'||NomeAutore||'~'
  from BIBLIOTECA_AUTORE
 order by Titolo

spool e:\oracle\external\book_auth_dump.lst
/
spool off
```

Oltre ai dati, i file di output conterranno anche una singola linea iniziale con il simbolo “/” e una linea finale che riporta “SQL> spool off”. Per semplificare la gestione dei dati, si dovrebbe modificare il file manualmente a livello di sistema operativo per eliminare queste linee di troppo.

Se un altro utente deve poter accedere ai dati contenuti nei file bookshelf\_dump.lst e book\_auth\_dump.lst, gli si dovrà concedere il privilegio READ nella directory LIBRO\_DIR:

```
grant read on directory LIBRO_DIR to altro_utente;
```

e i file stessi dovranno essere leggibili da parte dell’utente Oracle a livello di sistema operativo.

## 25.2 Creazione di una tabella esterna

Ora che i dati esterni sono disponibili e accessibili, è possibile creare la struttura di una tabella in grado di accedervi. Per crearla, si dovrà utilizzare la clausola organization external del comando create table, nella quale è possibile specificare la struttura di dati quasi come si farebbe per un control file di SQL\*Loader. Il listato seguente mostra la creazione della tabella BIBLIOTECA\_EXT, che si basa sui dati contenuti nel file di spool bookshelf.lst creato nel paragrafo precedente:

```
set feedback on heading on newpage 1 pagesize 60

create table BIBLIOTECA_EXT
(Titolo      VARCHAR2(100),
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Classificazione    VARCHAR2(2)
)
```

```

organization external
(type ORACLE_LOADER
default directory LIBRO_DIR
access parameters (records delimited by newline
fields terminated by "~"
    (Titolo      CHAR(100),
     Editore    CHAR(20),
     NomeCategoria CHAR(20),
     Classificazione   CHAR(2)
    ))
location ('bookshelf_dump.lst')
);

```

Oracle risponderà con:

Table created.

anche se nel database Oracle non saranno stati creati dei dati.

Analogamente, è possibile creare una tabella basata sul file di spool book\_auth\_dump.lst:

```

create table BIBLIOTECA_AUTORE_EXT
(Titolo      VARCHAR2(100),
NomeAutore  VARCHAR2(50)
)
organization external
(type ORACLE_LOADER
default directory LIBRO_DIR
access parameters (records delimited by newline
fields terminated by "~"
    (Titolo      CHAR(100),
     NomeAutore CHAR(50)
    ))
location ('book_auth_dump.lst')
);

```

**NOTA** Quando si crea la tabella esterna, Oracle eseguirà solamente delle convallide sommarie. La maggior parte degli errori non sarà visibile fino a quando non si cercherà di eseguire la query della tabella. La sintassi per i parametri di accesso è molto specifica e degli errori anche non gravi nella definizione dell'accesso potrebbero impedire l'accesso a tutte le righe.

È possibile verificare il contenuto delle tabelle esterne eseguendo una query partendo dalle medesime e confrontandole con le tabelle di origine, come mostrato nel listato seguente:

```
select Titolo from BIBLIOTECA
where NomeCategoria = 'ILLUSRAGAZZI';
```

TITOLO	-----
GOOD DOG, CARL	
POLAR EXPRESS	
RUNAWAY BUNNY	

3 rows selected.

```
select Titolo from BIBLIOTECA_EXT
where NomeCategoria = 'ILLUSRAGAZZI';
```

TITOLO

```
-----  
GOOD DOG, CARL  
POLAR EXPRESS  
RUNAWAY BUNNY
```

3 rows selected.

```
select COUNT(*) from BIBLIOTECA_AUTORE;
```

COUNT(\*)

```
-----  
37
```

```
select COUNT(*) from BIBLIOTECA_AUTORE_EXT;
```

COUNT(\*)

```
-----  
37
```

È possibile unire tramite join la tabella “interna” BIBLIOTECA\_AUTORE al suo equivalente esterno, BIBLIOTECA\_AUTORE\_EXT, per controllare che nessuna riga sia stata tralasciata o aggiunta:

```
select * from BIBLIOTECA_AUTORE BA
where not exists
(select 'x' from BIBLIOTECA_AUTORE_EXT BAE
 where BA.Titolo = BAE.Titolo
 and BA.NomeAutore = BAE.NomeAutore);
```

no rows selected

La tabella BIBLIOTECA\_AUTORE\_EXT punta al file book\_auth\_dump.lst. Se si modificano i dati nel file, anche i dati di BIBLIOTECA\_AUTORE\_EXT cambieranno. Come illustrato qui, è possibile eseguire la query delle tabelle esterne nello stesso modo in cui si esegue per le tabelle standard, in join, come parte di viste e così via. È possibile eseguire le funzioni sulle colonne della tabella esterna durante le query così come si farebbe per le tabelle standard.

È possibile eseguire la query sulla vista del dizionario dati USER\_EXTERNAL\_TABLES per ottenere informazioni relative alle tabelle esterne, tra cui le definizioni di accesso e delle directory predefinite:

```
desc USER_EXTERNAL_TABLES
```

Nome	Null?	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
TYPE_OWNER		CHAR(3)
TYPE_NAME	NOT NULL	VARCHAR2(30)
DEFAULT_DIRECTORY_OWNER		CHAR(3)
DEFAULT_DIRECTORY_NAME	NOT NULL	VARCHAR2(30)
REJECT_LIMIT		VARCHAR2(40)

ACCESS_TYPE	VARCHAR2(7)
ACCESS_PARAMETERS	VARCHAR2(4000)

Per esempio, la tabella BIBLIOTECA\_AUTORE\_EXT utilizza LIBRO\_DIR come propria directory predefinita, come mostrato nel listato seguente:

```
select * from USER_EXTERNAL_TABLES
where Nome_Tabella = 'BIBLIOTECA_AUTORE_EXT';
```

TABLE_NAME	TYP	TYPE_NAME	DEF
DEFAULT_DIRECTORY_NAME	REJECT_LIMIT		ACCESS_
ACCESS_PARAMETERS			
BOOKSHELF_AUTHOR_EXT	SYS	ORACLE_LOADER	SYS
BOOK_DIR	0		CLOB
records delimited by newline			
	fields terminated by "~"		
	(Titolo CHAR(100),		
	NomeAutore CHAR(50)		
	)		

USER\_EXTERNAL\_TABLES non mostra il nome del file esterno (o dei file) cui fa riferimento la tabella. Per visualizzare questa informazione, si esegua la query su USER\_EXTERNAL\_LOCATIONS:

```
select * from USER_EXTERNAL_LOCATIONS;
```

TABLE_NAME
LOCATION
DIR DIRECTORY_NAME
BIBLIOTECA_AUTORE_EXT
book_auth_dump.1st
SYS BOOK_DIR
BIBLIOTECA_EXT
bookshelf_dump.1st
SYS BOOK_DIR

## Opzioni per la creazione di tabelle esterne

Nella clausola organization external sono contenute quattro sottoclassi principali: type, default directory, access parameters e location. Quando si crea una tabella esterna, si possono utilizzare queste clausole per personalizzare il modo in cui Oracle visualizza i dati esterni.

### Type e Default Directory

La sintassi per il componente type è:

```
( [type tipo_driver_accesso] proprietà_dati_esterni )
[reject limit { intero | unlimited }]
```

Per le tabelle esterne, il driver di accesso è l'API utilizzata per trasformare i dati esterni. Per le tabelle esterne, si utilizzi il componente type ORACLE\_LOADER, l'unico tipo disponibile al momento, come mostrato negli esempi precedenti di questo capitolo.

**NOTA** *Dato che il driver di accesso fa parte del software Oracle, solamente ai file che sono accessibili dal database si potrà accedere come tabelle esterne. I file cui gli utenti Oracle non possono accedere non potranno essere utilizzati come tabelle esterne.*

Dopo la dichiarazione di type è possibile impostare un valore “reject limit”. Per default, nessuna riga può essere rifiutata, qualsiasi tipo di problema con una riga qualunque comporterà la restituzione da parte dell'istruzione select di un errore. Si crei un'altra copia dei dati di BIBLIOTECA in un file separato, e questa volta si dovranno lasciare nel file le linee aggiuntive che SQL\*Plus inserisce durante l'operazione spool:

```
set pagesize 0 newpage 0 feedback off
select Titolo||'~'||Editore||'~'||NomeCategoria||'~'||Classificazione||'~'
  from BIBLIOTECA
 order by Titolo

spool e:\oracle\external\bookshelf_dump_2.lst
/
spool off
```

Ora, si crei una nuova tabella che fa riferimento a questo file di spool, comunicando a Oracle di saltare il primo record (skip 1) e di consentire un altro errore (reject limit 1). Ciò spiega il simbolo “/” nella prima linea, e “SQL spool off” nell'ultima linea:

```
set feedback on heading on newpage 1 pagesize 60

create table BIBLIOTECA_EXT_2
(Titolo      VARCHAR2(100),
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Classificazione    VARCHAR2(2)
)
organization external
(type ORACLE_LOADER
 default directory LIBRO_DIR
 access parameters (records delimited by newline
skip 1
          fields terminated by
          "~-"
          (Titolo      CHAR(100),
           Editore     CHAR(20),
           NomeCategoria CHAR(20),
           Classificazione  CHAR(2)
          ) )
location ('bookshelf_dump_2.lst')
)
reject limit 1
;
```

Ora, si potrà verificare il numero di righe nella tabella:

```
set feedback on heading on newpage 1 pagesize 60
select COUNT(*) from BIBLIOTECA_EXT_2;
```

```
COUNT(*)
```

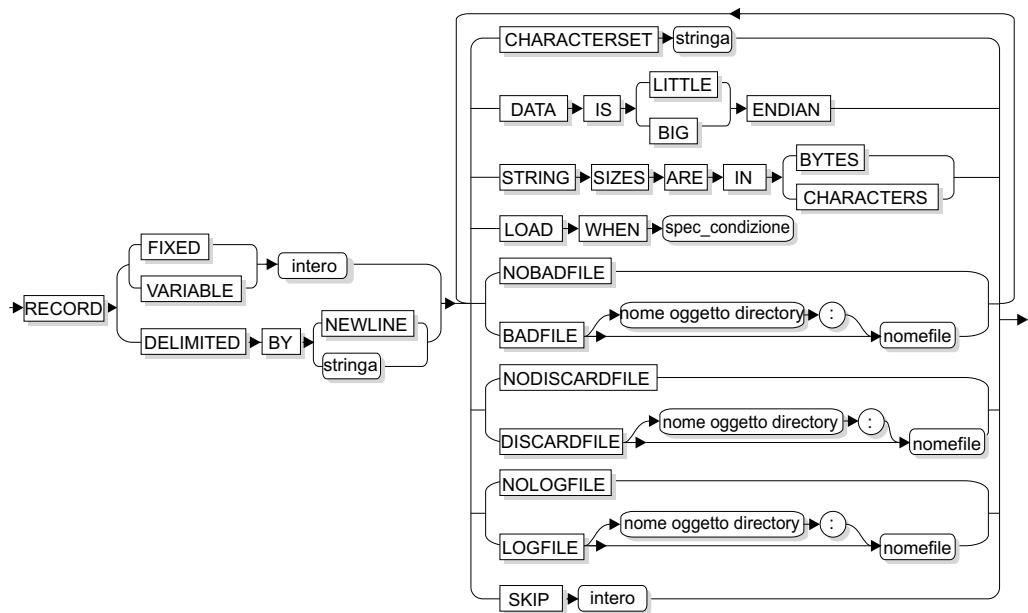
```
-----
```

```
31
```

La clausola default directory specifica quale oggetto di directory utilizzare per tutti quei file di dati che non specificano un'altra directory. Se si usano più file esterni collocati in più directory, si potrà chiamare una di queste come directory predefinita e specificare le altre con nomi di directory nella clausola location. Nella clausola location si dovranno usare nomi di oggetti di directory (come LIBRO\_DIR), e non il nome di percorso della directory completo.

### Access Parameters

La clausola access parameters comunica a Oracle come associare le righe contenute nel file alle righe contenute nella tabella. La sua sintassi è mostrata nell'illustrazione seguente:



Nella clausola access parameters per prima cosa si comunica a Oracle come creare un record, se la sua lunghezza è fissa o variabile, e poi come sono delimitate le righe. Nel caso dell'esempio di BIBLIOTECA\_EXT, i record sono delimitati da nuove linee. Se in una linea singola ci fossero più righe, si potrebbe usare una stringa di caratteri come separatore per le varie righe. Poiché i dati esterni potrebbero arrivare da un database non Oracle, Oracle supporta più set di caratteri e più dimensioni di stringa.

Come accadeva con SQL\*Loader, esiste la possibilità di specificare una clausola when per limitare le righe da selezionare. Nel listato seguente, viene creata la tabella BIBLIOTECA\_EXT\_3, con una clausola when (scritta in grassetto) per limitarla ai soli libri contenuti nella categoria ILLUSRAGAZZI.

```

create table BIBLIOTECA_EXT_3
(Titolo      VARCHAR2(100),
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Classificazione   VARCHAR2(2)
)
organization external
(type ORACLE_LOADER
default directory LIBRO_DIR
access parameters (records delimited by newline
load when CategoryName = 'ILLUSRAGAZZI'
skip 1
fields terminated by "~"
(Titolo      CHAR(100),
Editore     CHAR(20),
NomeCategoria CHAR(20),
Classificazione   CHAR(2)
)
)
location ('bookshelf_dump_2.lst')
)
reject limit 1
;

```

Ecco il risultato:

```

select SUBSTR(Titolo, 1,30), NomeCategoria
from BIBLIOTECA_EXT_3;

```

SUBSTR(TITOL0,1,30)	NOMECATEGORY
GOOD DOG, CARL	ILLUSRAGAZZI
POLAR EXPRESS	ILLUSRAGAZZI
RUNAWAY BUNNY	ILLUSRAGAZZI

3 rows selected.

BIBLIOTECA\_EXT\_3 accede allo stesso file di BIBLIOTECA\_EXT\_2, tuttavia mostra solo i record per la categoria ILLUSRAGAZZI a causa della clausola load when.

Come accadeva con SQL\*Loader, esiste la possibilità di creare un file di log, un file non valido e un file di scarto. Le righe che non rispettano la condizione load when verranno scritte nel file di scarto. Le righe che non rispettano le condizioni access parameters verranno scritte nel file non valido, mentre i dettagli di caricamento verranno scritti nel file di log. Per tutti e tre i tipi di file, è possibile specificare un oggetto di directory insieme al nome file in modo che sia possibile scrivere l'output in una directory diversa da quella del file di dati di input. È possibile specificare nodiscardfile, nobadfile e nologfile per impedire la creazione di questi file. Occorre utilizzare i nomi di oggetti di directory (come LIBRO\_DIR negli esempi riportati in questo capitolo) quando si specificano le posizioni dei file di scarto, dei file non validi e dei file di log. Se non si specificano delle posizioni per i file di log, i file non validi e i file di scarto, Oracle li creerà nella directory predefinita con nomi generati dal sistema.

Nella clausola access parameters si dovranno specificare anche definizioni e delimitatori di campo, come:

```

fields terminated by "~"
(Titolo      CHAR(100),
Editore     CHAR(20),

```

---

```
NomeCategoria    CHAR(20),
Classificazione CHAR(2))
```

Per impostare i valori per i valori di colonna NULL si può ricorrere alla clausola **missing field values are null**, tuttavia occorre prestare molta attenzione quando si usa questa opzione. Per esempio, la tabella AUTORE ha dei valori NULL nella colonna Commenti. La creazione di una tabella esterna per AUTORE\_EXT è mostrata nel listato seguente:

```
set pagesize 0 newpage 0 feedback off
select NomeAutore||'~'||Commenti||'~'
  from AUTORE
 order by NomeAutore

spool e:\oracle\external\author_dump.lst
/
spool off

set feedback on heading on newpage 1 pagesize 60
create table AUTORE_EXT
(NomeAutore  VARCHAR2(50),
 Commenti    VARCHAR2(100)
)
organization external
(type ORACLE_LOADER
 default directory LIBRO_DIR
 access parameters (records delimited by newline
                     skip 1
                     fields terminated by "~"
missing field values are null
                     (NomeAutore  CHAR(50),
                      Commenti    CHAR(100)
                     ) )
 location ('author_dump.lst')
)
reject limit 1
;
```

Tuttavia ciò non è corretto: se si selezionano i valori di NomeAutore da AUTORE\_EXT, si noterà che i valori includono:

```
select NomeAutore from AUTORE_EXT
  where NomeAutore like 'S%';
```

```
NOMEAUTORE
-----
SOREN KIERKEGAARD
STEPHEN AMBROSE
STEPHEN JAY GOULD
SQL> spool off
```

4 rows selected.

A causa della presenza della clausola **missing field values are null**, la linea “SQL> spool off” alla fine del listato è stata letta come un valore di NomeAutore, con un valore NULL per la colonna Commenti. Ciò evidenzia il problema legato alle eccezioni di codifica nelle definizioni

di SQL\*Loader: occorre essere assolutamente certi di aver capito bene i dati di origine e il modo in cui SQL\*Loader li tratterà. Nella maggior parte dei casi l'integrità dei dati verrà garantita meglio costringendo le righe a fallire (in file non validi o di scarto) e valutando le righe fallite separatamente dai caricamenti generali.

Per la sintassi completa disponibile per la clausola access parameters, fare riferimento alla voce SQL\*Loader nel Capitolo 42.

### **Location**

Nella clausola location si specificano i file di dati da utilizzare come dati di origine per la tabella. Nella clausola location è possibile nominare più file se tutti esistono in oggetti di directory per i quali l'utente possiede il privilegio READ. L'esempio seguente associa due file di spool di BIBLIOTECA separati per illustrare la capacità di combinare più file in un'unica tabella esterna.

```
create table BIBLIOTECA_EXT_4
(Titolo      VARCHAR2(100),
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Classificazione   VARCHAR2(2)
)
organization external
(type ORACLE_LOADER
 default directory LIBRO_DIR
access parameters (records delimited by newline
skip 1
fields terminated by "~"
(Titolo      CHAR(100),
Editore     CHAR(20),
NomeCategoria CHAR(20),
Classificazione   CHAR(2)
)
)
location ('bookshelf_dump_2.lst', 'bookshelf_dump.lst')
)
reject limit 1
:
```

L'ordine dei file è importante: skip 1 si applica al primo file e non al secondo. Il secondo file, bookshelf\_dump.lst, è il file che è stato modificato in precedenza per eliminare le righe non di dati nella prima e nell'ultima riga. Il risultato, che riproduce le righe in entrambi, è visibile nel listato seguente:

```
select COUNT(*) from BIBLIOTECA_EXT_4;
COUNT(*)
-----
62
```

### **25.3 Limitazioni, vantaggi e usi possibili delle tabelle esterne**

Le tabelle esterne sono soggette a limitazioni che potrebbero renderle inadatte per alcune applicazioni per l'elaborazione di transazioni in linea. Nelle tabelle esterne non si possono effettuare operazioni di inserimento, aggiornamento o cancellazione. Più dinamica è una tabella e meno

appropriati potrebbero essere i file esterni. Come evidenziano gli esempi proposti in questo capitolo, è possibile modificare il file in modo dinamico a livello del sistema operativo. Se l'applicazione genera solamente degli inserimenti, si potrebbe avere la possibilità di scrivere questi record inseriti in un file esterno invece che in una tabella di database.

Le tabelle esterne non possono essere indicizzate. La mancanza di indici nelle tabelle esterne non deve essere per forza un fattore negativo per le prestazioni dell'applicazione. Le query delle tabelle esterne si completano molto rapidamente, anche se per ogni accesso è richiesta la scansione intera di una tabella. È coinvolto anche il sistema di I/O, tuttavia i sistemi di I/O moderni utilizzano le tecniche di caching e RAID per ridurre in modo significativo il calo di prestazioni associato a scansioni intere ripetute dello stesso file.

In una tabella esterna non è possibile specificare dei vincoli. Persino il tentativo di creare un vincolo NOT NULL o quello di una chiave esterna avrà esito negativo:

```
alter table BIBLIOTECA_EXT add constraint CATFK
foreign key (NomeCategoria) references CATEGORIA(NomeCategoria);

foreign key (NomeCategoria) references CATEGORIA(NomeCategoria)
*
ERROR at line 2:
ORA-30657: operation not supported on external organized table
```

Nonostante queste limitazioni, le tabelle esterne mettono a disposizione molte soluzioni utili: possono essere unite tramite join (una all'altra, oppure a tabelle standard), si può ricorrere ai suggerimenti per costringere l'ottimizzatore a scegliere percorsi di join differenti, e i risultati saranno visibili nei percorsi di esecuzione delle query (per dettagli sui suggerimenti e sull'ottimizzatore di Oracle, fare riferimento al Capitolo 38).

Come alternativa al caricamento dei dati, le tabelle esterne offrono ai DBA e agli sviluppatori di applicazioni la possibilità di accedere ai dati senza dover supportare programmi di caricamento a lunga esecuzione. Dato che i file possono essere modificati a livello del sistema operativo, è possibile sostituire i dati di una tabella molto rapidamente senza doversi preoccupare delle transazioni in sospeso che modificano la tabella. Per esempio, si potrebbe sfruttare questa capacità per creare più tabelle esterne e un'unica vista union all tra tutte queste tabelle, dando così origine a una vista di partizione tra più file. Quindi, si potranno gestire i dati di ogni tabella separatamente a livello del file system, sostituendone il contenuto in base alle esigenze.

Considerata la possibilità di eseguire una query su una tabella esterna, quest'ultima potrà fungere da origine di dati per un comando `insert as select`. Durante questa operazione, Oracle cercherà di caricare i file esterni in parallelo, allo scopo di migliorare le prestazioni. Per migliorare ulteriormente le prestazioni dell'operazione `insert as select`, si dovrebbe usare il suggerimento APPEND per forzare gli inserimenti a livello di blocco. Quando si specifica il grado di parallelismo per l'operazione `insert as select`, Oracle avvia più driver di accesso ORACLE\_LOADER per elaborare i dati in parallelo. Per migliorare ulteriormente le prestazioni delle operazioni di caricamento, è consigliabile evitare di utilizzare campi a lunghezza variabile, campi delimitati, conversioni di set di caratteri, NULLIF, DEFAULTIF e operazioni di conversione per i tipi di dati. La disattivazione di badfile (con nobadfile) elimina i costi associati alla creazione di file e alla gestione del contesto della riga originale.

Durante l'operazione `insert as select` si possono eseguire funzioni sui dati mentre questi vengono elaborati. Queste funzioni possono essere eseguite nella sintassi del comando `insert as select` oppure nella definizione della tabella esterna. Questa capacità sottolinea un vantaggio importante garantito dalle tabelle esterne, ossia la possibilità di centralizzare i requisiti di rappresentazione ed elaborazione dei dati, creando così le routine di conversione nelle definizioni delle tabelle. Nei control file SQL\*Loader o nelle routine PL/SQL non ci sono dati di elabora-

zione memorizzati; tutta la logica è costruita nella definizione della tabella, accessibile tramite `USER_EXTERNAL_TABLES`.

Durante le query, le tabelle esterne consentono di selezionare gruppi di dati specifici (tramite la clausola `load when`, come mostrato in questo capitolo). Se per il caricamento di una data warehouse si hanno più sorgenti di dati, si potrà scegliere quali dati mettere a disposizione anche quando questi sono esterni al database. Questa soluzione può servire per mantenere la disponibilità dell'applicazione durante i caricamenti di dati. Queste operazioni di caricamento possono essere eseguite in parallelo, se il file esterno presenta un formato di file `FIXED`.

Inoltre, la soluzione di accesso limitato consente anche di rafforzare le regole di sicurezza complesse che riguardano l'accesso ai dati. Per esempio, si potrebbe tenere i dati riservati fuori dal database, in una directory sicura. Gli utenti che possiedono l'accesso `READ` a questa directory potrebbero essere in grado di usare la tabella esterna e unirla alle altre tabelle; gli utenti privi di questo accesso potrebbero accedere solamente a quei dati inseriti in tabelle pubblicamente accessibili. I dati altamente sicuri, o i dati dinamici ad accesso limitato, non dovranno essere inseriti nel database a meno che non sia necessario.

Se nell'architettura del database si utilizzano tabelle esterne, ci si dovrà accertare che i piani di backup e recupero tengano conto di questi file così come del resto del database. Se i file esterni cambiano più rapidamente di quelli del database, potrebbe essere necessario eseguire dei backup con maggior frequenza per sfruttare le capacità di recupero completo di Oracle.

## Capitolo 26

# Uso delle query flashback

- 26.1 **Esempi di flashback basati sull'ora**
- 26.2 **Come salvare i dati**
- 26.3 **Esempi di flashback basati su SCN**

Come parte del proprio modello di coerenza in lettura, Oracle visualizza i dati che sono stati destinati al database in seguito a transazioni confermate con un comando `commit`. A partire da Oracle9i è possibile eseguire query dei dati così com'erano prima che venisse eseguita una transazione. Se accidentalmente si esegue un aggiornamento o una cancellazione errati, si potrà ricorrere a questa capacità, chiamata *query flashback*, per visualizzare i dati com'erano prima dell'esecuzione. I risultati della query flashback potranno servire per ripristinare i dati.

Le query flashback sono limitate. Per usarle, il database dovrà implementare gli undo gestiti a livello di sistema, una caratteristica introdotta con Oracle9i in sostituzione dei segmenti di rollback; per verificare se questa caratteristica è stata abilitata nel proprio ambiente, si consulti il DBA, il quale dovrà creare una tablespace di undo, abilitare la gestione automatica degli undo e stabilire una finestra con il tempo di conservazione degli undo. Oracle cercherà di mantenere informazioni di undo sufficienti nella tablespace di undo allo scopo di supportare le query flashback durante il periodo di conservazione degli undo. L'impostazione del tempo di conservazione e la quantità di spazio disponibile nella tablespace di undo possono influire significativamente sulla capacità di eseguire con successo una query flashback.

**NOTA** Oracle usa gli undo per eseguire il rollback delle transazioni e per supportare le query flashback. Oracle usa i redo (catturati nei redo log file in linea) per applicare le transazioni durante i recuperi del database.

Come si potrà osservare negli esempi di questo capitolo, le query flashback non sono semplici. Benché i miglioramenti apportati alla sintassi sono stati documentati in quanto inclusi in Oracle9i Release 2, le query flashback richiedono ancora comandi SQL non standard. Quando si progetta un'applicazione, le query flashback non dovrebbero essere considerate come parte del progetto a causa della loro complessità, dell'impossibilità di prevederne le prestazioni e per la loro dipendenza da elementi di sistema che esulano dal controllo dello sviluppatore dell'applicazione (come il numero di transazioni eseguite in un certo lasso di tempo e la dimensione delle tablespace di undo). Al contrario, le si dovrebbe trattare come un'opzione durante fasi difficili di recupero, supporto e verifica dei dati. Per esempio, potrebbero servire per creare copie di tabelle in più punti ormai passati in modo da utilizzarle per ricostruire i dati modificati.

**NOTA** Per usare le query flashback, è necessario disporre del privilegio EXECUTE nel package DBMS\_FLASHBACK.

## 26.1 Esempi di flashback basati sull'ora

La tabella LIBRO\_ORDINE ha sei record, come mostrato nel listato seguente:

```
select * from LIBRO_ORDINE;
```

TITOLO	EDITORE	NOME CATEGORIA
SHOELESS JOE	MARINER	NARRADULTI
GOSPEL	PICADOR	NARRADULTI
SOMETHING SO STRONG	PANDORAS	SAGGIADULTI
GALILEO'S DAUGHTER	PENGUIN	SAGGIADULTI
LONGITUDE	PENGUIN	SAGGIADULTI
ONCE REMOVED	SANCTUARY PUB	SAGGIADULTI

6 rows selected.

Una spedizione è arrivata, quindi i vecchi record di LIBRO\_ORDINI vengono cancellati. Sfortunatamente non tutti i libri sono stati spediti, quindi la cancellazione non è appropriata:

```
delete from LIBRO_ORDINE;
commit;
```

Come si possono ricostruire i record non ricevuti partendo dalla tabella LIBRO\_ORDINE, dato che si possiedono solo i libri ricevuti? Si potrebbe eseguire un recupero del database utilizzando import per ripristinare la tabella, oppure eseguendo un recupero del database fisico. Le query flashback consentono di evitare l'esecuzione di tutte queste operazioni di recupero.

Innanzitutto, si esegua la query sui dati vecchi partendo dal database. Dato che la modalità flashback supporta soltanto le query, rimanderemo l'analisi di ulteriori operazioni sui dati (come il loro salvataggio in una tabella separata) a un esempio più complesso del paragrafo successivo. Per entrare nella modalità flashback, occorre utilizzare la procedura ENABLE\_AT\_TIME del package DBMS\_FLASHBACK. Nell'esempio seguente, il flashback viene attivato a un istante passato da cinque minuti (SysDate-5/1440).

```
select COUNT(*) from LIBRO_ORDINE;

COUNT(*)
-----
0
```

```
execute DBMS_FLASHBACK.ENABLE_AT_TIME(SysDate-5/1440);
```

PL/SQL procedure successfully completed.

```
select COUNT(*) from LIBRO_ORDINE;

COUNT(*)
-----
6
```

Dopo aver completato la query, si dovrà disattivare la modalità flashback:

```
execute DBMS_FLASHBACK.DISABLE;

PL/SQL procedure successfully completed.
```

**NOTA** Prima di abilitare ancora la modalità flashback, la si dovrà disattivare.

Quando si esegue una query flashback, solo lo stato dei dati subisce una modifica. Si utilizzano l'ora di sistema corrente (come si può notare eseguendo la query del valore SysDate) e il dizionario dei dati corrente.

**NOTA** Non è possibile eseguire una query flashback di una tabella remota tramite un database link.

## 26.2 Come salvare i dati

Come mostrato nell'esempio di LIBRO\_ORDINE, le query flashback sono facili da implementare, a patto che la gestione degli undo del database sia stata configurata correttamente e le informazioni di undo siano disponibili. Ma come si possono usare i dati flashback? Mentre ci si trova nella modalità flashback, non è possibile eseguire operazioni DDL o DML, quindi non si potrà eseguire un comando create table as select per salvare i dati vecchi in una tabella di lavoro temporanea.

Per eseguire operazioni DDL e DML su dati flashback, si dovrà ricorrere a PL/SQL, l'estensione di linguaggio procedurale di Oracle al linguaggio di programmazione SQL (fare riferimento al Capitolo 27). La sequenza dei comandi per il linguaggio PL/SQL utilizzato per salvare i dati flashback è simile alla seguente:

1. Definire il cursore che esegue la query sui dati e una variabile per contenere i dati.
2. Iniziare il blocco PL/SQL.
3. Attivare la modalità delle query flashback.
4. Aprire il cursore.
5. Disattivare la modalità delle query flashback.
6. Eseguire ciclicamente i record selezionati, uscendo quando non ci sono più record.

A questo punto occorre salvare i vecchi dati. Mentre la modalità flashback è disattivata, si crei una nuova tabella che abbia la medesima struttura di LIBRO\_ORDINE, ma priva di record:

```
drop table LIBRO_ORDINE_FLASH;
create table LIBRO_ORDINE_FLASH
(Titolo      VARCHAR2(100) primary key,
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20));
```

Table created.

Ora, si esegua un blocco PL/SQL. Nella definizione del cursore, si possono specificare le righe e le colonne che si intendono restituire. In questo esempio, viene selezionata l'intera tabella.

```
declare cursor VECCHIO_LIBRO_ORDINE is select * from LIBRO_ORDINE;
      VLO    LIBRO_ORDINE%ROWTYPE;
begin
  DBMS_FLASHBACK.ENABLE_AT_TIME(SysDate-30/1440);
  open VECCHIO_LIBRO_ORDINE;
  DBMS_FLASHBACK.DISABLE;
```

```

loop
  fetch VECCHIO_LIBRO_ORDINE into VLO;
  exit when VECCHIO_LIBRO_ORDINE%NOTFOUND;
  insert into LIBRO_ORDINE_FLASH
    values (VLO.Titolo, VLO.Editore, VLO.NomeCategoria);
end loop;
close VECCHIO_LIBRO_ORDINE;
commit;
end;
/

```

Oracle risponde con:

PL/SQL procedure successfully completed.

Ora, si può controllare se i dati flashback sono stati salvati:

```

column Titolo format a30
select * from LIBRO_ORDINE_FLASH;

TITOLO                EDITORE          NOMECATEGORYIA
-----                -----
SHOELESS JOE           MARINER          NARRADULTI
GOSPEL                 PICADOR          NARRADULTI
SOMETHING SO STRONG   PANDORAS        SAGGIADULTI
GALILEO'S DAUGHTER    PENGUIN          SAGGIADULTI
LONGITUDE              PENGUIN          SAGGIADULTI
ONCE REMOVED           SANCTUARY PUB   SAGGIADULTI

```

6 rows selected.

Se la query flashback cercasse di usare un indice creato dopo il punto di flashback, non riuscirebbe. In questo caso, si dovrebbero usare i suggerimenti per comunicare all'ottimizzatore quale metodo di accesso ai dati utilizzare (una scansione intera di tabella oppure un indice diverso); per dettagli sull'impiego dei suggerimenti, si consulti il Capitolo 38.

## Limitazioni ai flashback basati sull'ora

Gli esempi precedenti hanno mostrato come utilizzare l'indicatore orario di sistema durante una query flashback. Va detto però che Oracle internamente usa il System Change Number (SCN), e non l'indicatore orario di sistema, per creare i dati di flashback. L'indicatore orario di sistema viene aggiornato ai valori SCN ogni cinque minuti; la tabella SYS.SMON\_SCN\_TIME mostra almeno gli ultimi cinque giorni di corrispondenze tra indicatore orario di sistema e SCN. Quando si torna indietro a un certo punto nel tempo, Oracle usa l'indicatore orario di cinque minuti più recente che precede il momento richiesto. Se SMON\_SCN\_TIME ha un record per 1:00 P.M. e 1:05 P.M., una richiesta di un flashback a 1:04 P.M. restituirà i dati con orario 1:00 P.M.

SMON\_SCN\_TIME mantiene solamente le 1.440 voci più recenti, pertanto le query flashback che vanno oltre i cinque giorni più recenti in un database sempre aperto dovranno utilizzare un approccio basato su SCN, come descritto nel prossimo paragrafo. Se si chiude il database periodicamente (come in occasione dei backup fuori linea), i 1.440 record in SMON\_SCN\_TIME supereranno i cinque giorni.

## 26.3 Esempi di flashback basati su SCN

Quando si eseguono flashback basati sull'ora, in realtà si effettuano flashback basati su SCN, facendo affidamento su Oracle affinché trovi un SCN vicino all'orario specificato. Se si conosce il valore SCN esatto, si potrà eseguire un flashback con un alto grado di perfezione. Al posto della procedura ENABLE\_AT\_TIME, si chiamerà la procedura ENABLE\_AT\_SYSTEM\_CHANGE\_NUMBER, come mostrato nell'esempio seguente.

Per iniziare un flashback basato su SCN, occorre innanzi tutto conoscere il valore SCN della transazione. Per ottenere il numero dell'ultima modifica, si esegua prima un comando commit e poi la funzione GET\_SYSTEM\_CHANGE\_NUMBER del package DBMS\_FLASHBACK prima di eseguire la transazione. L'esempio seguente mostra questo processo come parte di una transazione sulla tabella LIBRO\_ORDINE\_FLASH creata e popolata nella prima parte di questo capitolo. Per prima cosa il valore SCN corrente viene assegnato a una variabile di nome SCN\_FLASH e visualizzato tramite il comando print di SQL\*Plus:

```
commit;
variable SCN_FLASH number;
execute :SCN_FLASH :=DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
```

PL/SQL procedure successfully completed.

```
print SCN_FLASH
```

```
SCN_FLASH
-----
1589270
```

Quindi viene eseguito il comando delete e il risultato viene visualizzato:

```
delete from LIBRO_ORDINE_FLASH;
```

6 rows deleted.

```
commit;
```

Ora, si può eseguire la query dei dati flashback. Anche se il valore di SCN è noto, se si è ancora nella medesima sessione si può continuare a usare il valore della variabile SCN\_FLASH:

```
execute DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER(:SCN_FLASH);
```

PL/SQL procedure successfully completed.

```
select COUNT(*) from LIBRO_ORDINE_FLASH;
```

```
COUNT(*)
-----
6
```

```
execute DBMS_FLASHBACK.DISABLE;
```

Ora, si possono usare i dati flashback contenuti in LIBRO\_ORDINE\_FLASH, cui si accede con la procedura ENABLE\_AT\_SYSTEM\_CHANGE\_NUMBER per popolare LIBRO\_ORDINE:

```

declare cursor VECCHIO_LIBRO_ORDINE is select * from LIBRO_ORDINE_FLASH;
      VLO_LIBRO_ORDINE_FLASH%ROWTYPE;
begin
  DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER(:SCN_FLASH);
  open VECCHIO_LIBRO_ORDINE;
  DBMS_FLASHBACK.DISABLE;
  loop
    fetch VECCHIO_LIBRO_ORDINE into VLO;
    exit when VECCHIO_LIBRO_ORDINE%NOTFOUND;
    insert into LIBRO_ORDINE
      values (VLO.Titolo, VLO.Editore, VLO.NomeCategoria);
  end loop;
  close VECCHIO_LIBRO_ORDINE;
  commit;
end;
/

```

PL/SQL procedure successfully completed.

```
select COUNT(*) from LIBRO_ORDINE;
```

```
COUNT(*)
```

```
-----
```

```
6
```

**NOTA** *Le operazioni DDL che alterano la struttura di una tabella invalidano i vecchi dati degli undo della tabella, e le capacità di flashback sono limitate al momento da cui la DDL è stata eseguita. Le modifiche legate allo spazio (come la modifica di pctfree) non invalidano i vecchi dati degli undo.*

Per i DBA, le query flashback potrebbero rappresentare uno strumento per evitare le operazioni di recupero. Se i dati possono essere ricostruiti grazie a query flashback, e se il volume dei dati non è enorme, si potrebbero salvare i risultati di più query flashback in tabelle separate. Quindi, si potrebbero mettere a confronto i dati contenuti nelle varie tabelle grazie alle opzioni SQL mostrate nei capitoli precedenti: subquery correlate, exists, not exists, minus e così via. Se non si riesce a evitare in nessun modo un'operazione di recupero, si dovrebbe comunque essere in grado di individuare il periodo di tempo da usare per un'operazione di recupero con criterio orario.

Agli sviluppatori e amministratori di applicazioni le query flashback offrono uno strumento importante per la ricostruzione dei dati. Le query flashback possono diventare particolarmente complicate durante le operazioni di verifica e di supporto. Invece di inserire le query flashback nella progettazione delle applicazioni di produzione, le si dovrebbe utilizzare come un'opzione di fallback per quei casi di assistenza che non possono essere risolti senza che i dati siano interessati.

Parte terza

## **PL/SQL**



## Capitolo 27

# Introduzione a PL/SQL

- 27.1 **PL/SQL: nozioni di base**
- 27.2 **La sezione delle dichiarazioni**
- 27.3 **La sezione dei comandi eseguibili**
- 27.4 **La sezione di gestione delle eccezioni**

PL/SQL è un linguaggio di query strutturato (SQL, Structured Query Language) che a sua volta ha come sottoinsieme il linguaggio procedurale PL (Procedural Language) di Oracle. Il linguaggio PL/SQL può essere utilizzato per codificare le regole business mediante la creazione di stored procedure e package, per attivare eventi di database quando necessario o per aggiungere una logica di programmazione all'esecuzione di comandi SQL.

I passaggi richiesti per la creazione di trigger, stored procedure e package sono descritti nei capitoli seguenti di questa parte del libro. In questo capitolo vengono invece descritte le strutture e la sintassi più semplici utilizzate nel linguaggio PL/SQL.

## 27.1 PL/SQL: nozioni di base

Il codice PL/SQL è raggruppato in strutture dette *blocchi*. Se si crea una stored procedure o un package, si assegna un nome al blocco di codice PL/SQL. Se al blocco di codice PL/SQL non viene assegnato un nome, verrà definito come *blocco anonimo*. Gli esempi riportati in questo capitolo contengono blocchi anonimi di codice PL/SQL. I prossimi capitoli di questa parte descriveranno la creazione di blocchi dotati di nome.

Un blocco di codice PL/SQL contiene tre sezioni:

SEZIONE	DESCRIZIONE
Dichiarazioni	Definisce e inizializza le variabili e i cursori utilizzati nel blocco.
Comandi eseguibili	Usa i comandi di controllo del flusso (come i comandi if e i cicli) per eseguire i comandi e assegnare dei valori alle variabili dichiarate.
Gestione delle eccezioni	Fornisce una gestione personalizzata delle condizioni di errore.

All'interno di un blocco PL/SQL, quella delle dichiarazioni è la prima sezione; in essa si definiscono le variabili e i cursori utilizzati dal blocco. La sezione delle dichiarazioni inizia con la parola chiave `declare` e termina quando inizia la sezione dei comandi eseguibili, indicata dalla parola chiave `begin`. Questa, a sua volta, è seguita dalla sezione di gestione delle eccezioni, il cui inizio viene indicato dalla parola chiave `exception`. Il blocco PL/SQL termina infine con la parola chiave `end`.

Di seguito è riportata la struttura di un tipico blocco PL/SQL:

```
declare
  <sezione dichiarazioni>
begin
  <comandi eseguibili>
exception
  <gestione delle eccezioni>
end;
```

Nei prossimi paragrafi di questo capitolo viene descritta ciascuna sezione del blocco PL/SQL.

## 27.2 La sezione delle dichiarazioni

La sezione delle dichiarazioni dà inizio a un blocco PL/SQL. Questa sezione inizia con la parola chiave **declare**, seguita da un elenco di definizioni di variabili e cursori. Le variabili possono essere definite con valori costanti e possono ereditare il tipo di dati da colonne già esistenti e risultati di query, come mostrato negli esempi seguenti.

Nel listato che segue viene calcolata l'area di un cerchio. Il risultato viene memorizzato in una tabella di nome AREE. Questa tabella ha due colonne per memorizzare i valori del raggio e quelli dell'area. L'area del cerchio viene calcolata elevando al quadrato il valore del raggio e moltiplicando il risultato per la costante *pi*:

```
declare
  pi    constant NUMBER(9,7) := 3.1415927;
  raggio INTEGER(5);
  area  NUMBER(14,2);
begin
  raggio := 3;
  area := pi*power(raggio,2);
  insert into AREE values (raggio, area);
end;
```

end segnala la fine del blocco PL/SQL; in base allo strumento utilizzato potrebbe essere necessario aggiungere una barra / per eseguire il blocco PL/SQL. Quando si esegue il blocco PL/SQL, Oracle genera questa risposta:

PL/SQL procedure successfully completed.

Per verificare che il blocco PL/SQL sia stato completato correttamente, è possibile selezionare dal database le righe che sono state inserite mediante il codice PL/SQL. La query e i risultati del listato seguente riportano la riga creata dal blocco PL/SQL nella tabella AREE.

```
select *
from AREE;

RAGGIO      AREA
----- -----
3           28.27
```

L'output mostra che il blocco PL/SQL ha inserito nella tabella AREE una singola riga. È stata creata una sola riga in quanto era stato specificato un solo valore per raggio.

Nella prima sezione del blocco PL/SQL, mostrata nel listato seguente, vengono dichiarate tre variabili. Qui si devono dichiarare le variabili che verranno utilizzate nella sezione dei comandi eseguibili del blocco PL/SQL.

```
declare
    pi      constant NUMBER(9,7) := 3.1415927;
    raggio INTEGER(5);
    area   NUMBER(14,2);
```

La prima variabile dichiarata è *pi*, impostata a un valore costante mediante la parola chiave **constant**. Dato che è stato dichiarato come costante, il valore della variabile *pi* non potrà essere modificato nella sezione dei comandi eseguibili. Il valore viene assegnato tramite l'operatore **:=**.

```
pi      constant NUMBER(9,7) := 3.1415927;
```

I valori costanti possono essere assegnati anche con la parola chiave **default**:

```
pi      NUMBER(9,7) DEFAULT 3.1415927;
```

Le due variabili successive sono definite, ma a esse non vengono assegnati valori predefiniti:

```
raggio INTEGER(5);
area   NUMBER(14,2);
```

Nella sezione delle dichiarazioni è possibile assegnare un valore iniziale a una variabile. Per impostare un valore iniziale per una variabile, occorre far seguire alla relativa specifica del tipo di dati l'assegnamento del valore, come mostrato nel listato seguente:

```
raggio INTEGER(5) := 3;
```

Nell'esempio, i tipi di dati comprendono **NUMBER** e **INTEGER**. I tipi di dati PL/SQL comprendono tutti di tipi di dati SQL validi e quelli complessi basati su strutture di query. La Tabella 27.1 elenca i tipi di dati PL/SQL supportati.

Nell'esempio seguente, viene dichiarato un cursore per recuperare un record dalla tabella **VALORI\_RAGGIO**. La tabella **VALORI\_RAGGIO** è costituita da una sola colonna, **Raggio**, che contiene i valori del raggio da utilizzare in questi esempi. Il cursore viene dichiarato nella sezione delle dichiarazioni. Inoltre, viene dichiarata una variabile di nome “*val\_rag*” con un tipo di dati basato sui risultati del cursore.

```
declare
    pi      constant NUMBER(9,7) := 3.1415927;
    area   NUMBER(14,2);
    cursore rag_cursore is
        select * from VALORI_RAGGIO;
    val_rag rag_cursore%ROWTYPE;
begin
    open rag_cursore;
    fetch rag_cursore into val_rag;
    area := pi*power(val_rag.raggio,2);
    insert into AREE values (val_rag.raggio, area);
close rag_cursore;
end;
```

Per questo esempio, la tabella **VALORI\_RAGGIO** contiene una sola riga con un valore di **Raggio** pari a 3.

**Tabella 27.1** Tipi di dati PL/SQL.

<b>TIPI DI DATI SCALARI</b>			
BINARY_INTEGER	INT	NVARCHAR2	STRING
BOOLEAN	INTEGER	NUMBER	TIMESTAMP
CHAR	INTERVAL DAY TO SECOND	NUMERIC	TIMESTAMP WITH LOCAL TIME ZONE
CHARACTER	INTERVAL YEAR TO MONTH	PLS_INTEGER	TIMESTAMP WITH TIME ZONE
DATE	LONG	POSITIVE	UROWID
DEC	LONG RAW	POSITIVEN	VARCHAR
DECIMAL	NATURAL	REAL	VARCHAR2
DOUBLE PRECISION	NATURALN	SIGNTYPE	RAW
FLOAT	NCHAR	SMALLINT	ROWID
<b>TIPI COMPOSTI</b>			
RECORD	TABLE	VARRAY	
<b>TIPI RIFERIMENTO</b>			
REF CURSOR	REF <i>tipo Oggetto</i>		
<b>TIPI LOB</b>			
BFILE	BLOB	CLOB	NCLOB

Nella prima parte della sezione delle dichiarazioni, vengono definite le variabili *pi* e *area*, come nei precedenti esempi del capitolo. La variabile “raggio” non è definita; al suo posto viene definito un cursore denominato “*rag\_cursore*”. La definizione del cursore è costituita da un nome per il cursore (“*rag\_cursore*”) e in una query (select \* from VALORI\_RAGGIO;). Un cursore conserva i risultati di una query affinché possano essere elaborati da altri comandi all’interno del blocco PL/SQL:

```
declare
  pi      constant NUMBER(9,7) := 3.1415927;
  area   NUMBER(14,2);
  cursore rag_cursore is
    select * from VALORI_RAGGIO;
```

Una dichiarazione finale crea una variabile la cui struttura viene ancorata dal gruppo di risultati del cursore:

```
val_rag rag_cursore%ROWTYPE;
```

La variabile “*val\_rag*” sarà in grado di fare riferimento a ciascuna colonna del gruppo di risultati della query. In questo esempio, la query restituisce solo una singola colonna, tuttavia se la tabella contenesse più colonne, sarebbe possibile fare riferimento a tutte queste mediante la variabile “*val\_rag*”.

Oltre alla dichiarazione %ROWTYPE, è possibile utilizzare la dichiarazione %TYPE per ereditare informazioni relative al tipo di dati. Se si utilizza la dichiarazione %ROWTYPE, la variabile eredita le informazioni sulla colonna e il tipo di dati per tutte le colonne contenute nel gruppo di risultati del cursore. Se invece si usa la dichiarazione %TYPE, la variabile eredita solamente la definizione della colonna utilizzata per definirla. Inoltre è possibile basare le definizioni %TYPE su cursori, come nell'esempio seguente:

```
cursor rag_cursore is
    select * from VALORI_RAGGIO;
val_rag rag_cursore%ROWTYPE;
raggio_val_rag val_rag.Raggio%TYPE;
```

Nel listato precedente, la variabile “*val\_rag*” eredita i tipi di dati del gruppo di risultati del cursore “*rag\_cursore*”. La variabile “*val\_rag\_raggio*” eredita il tipo di dati della colonna Raggio all'interno della variabile “*val\_rag*”.

Il vantaggio dell'ancoraggio dei tipi di dati con le definizioni %ROWTYPE e %TYPE è la possibilità di rendere le definizioni del tipo di dati nel codice PL/SQL indipendenti dalle strutture dei dati base. Se la colonna Raggio della tabella VALORI\_RAGGIO viene modificata dal tipo di dati NUMBER(5) al tipo di dati NUMBER(4,2), non sarà necessario modificare il codice PL/SQL. Il tipo di dati assegnato alle variabili associate verrà determinato dinamicamente durante l'esecuzione.

### 27.3 La sezione dei comandi eseguibili

Nella sezione dei comandi eseguibili vengono elaborati le variabili e i cursori dichiarati nella sezione delle dichiarazioni del blocco PL/SQL. Questa sezione inizia sempre con la parola chiave *begin*. Nel listato seguente viene ripetuto il primo esempio di blocco PL/SQL dalla sezione delle dichiarazioni: viene calcolata l'area di un cerchio e i risultati vengono inseriti nella tabella AREE:

```
declare
    pi      constant NUMBER(9,7) := 3.1415927;
    raggio INTEGER(5);
    area    NUMBER(14,2);
begin
    raggio := 3;
    area := pi*power(raggio,2);
    insert into AREE values (raggio, area);
end;
```

Nel listato precedente, la sezione dei comandi eseguibili è la seguente:

```
begin
    raggio := 3;
    area := pi*power(raggio,2);
    insert into AREE values (raggio, area);
end;
```

Dopo la parola chiave *begin* inizia il lavoro del blocco PL/SQL. Per prima cosa, viene assegnato un valore alla variabile “*raggio*”. Il valore della variabile “*raggio*” e della costante *pi* vengono utilizzati per determinare il valore della variabile “*area*”. I valori di Raggio e Area vengono quindi inseriti nella tabella AREE.

La sezione dei comandi eseguibili del blocco PL/SQL può contenere comandi che eseguono i cursori dichiarati nella sezione delle dichiarazioni. Nell'esempio seguente (tratto dal paragrafo dedicato alla sezione delle dichiarazioni), la sezione dei comandi eseguibili presenta numerosi comandi che fanno riferimento al cursore “*rag\_cursore*”:

```
declare
    pi      constant NUMBER(9,7) := 3.1415927;
    area   NUMBER(14,2);
    cursore rag_cursore is
        select * from VALORI_RAGGIO;
    val_rag rag_cursore%ROWTYPE;
begin
    open rag_cursore;
    fetch rag_cursore into val_rag;
    area := pi*power(val_rag.raggio,2);
    insert into AREE values (val_rag.raggio, area);
    close rag_cursore;
end;
```

Nel primo comando riguardante il cursore viene utilizzato `open`:

```
open rag_cursore;
```

Quando si apre il cursore “*rag\_cursore*” (tramite il comando `open`), la query dichiarata per questo cursore viene eseguita, e vengono identificati i record da restituire. Quindi, i record vengono estratti dal cursore con il comando `fetch`.

```
fetch rag_cursore into val_rag;
```

Nella sezione delle dichiarazioni, la variabile “*val\_rag*” era stata dichiarata in modo da ancorare i propri tipi di dati dal cursore “*rag\_cursore*”:

```
cursore rag_cursore is
    select * from VALORI_RAGGIO;
    val_rag rag_cursore%ROWTYPE;
```

Quando si estrae (con `fetch`) un record dal cursore per inserirlo nella variabile “*val\_rag*”, è ancora possibile indirizzare ciascun valore della colonna selezionato mediante la query del cursore. Quando i dati del cursore non sono più necessari, è possibile chiudere il cursore (con il comando `close`), come mostrato nel listato seguente.

```
close rag_cursore;
```

La sezione dei comandi eseguibili può contenere logica condizionale, come i comandi `if` e i cicli. Nei paragrafi successivi verranno riportati esempi per ciascuno dei tipi principali di operazioni per il controllo di flusso consentiti nel linguaggio PL/SQL. Le operazioni per il controllo del flusso possono essere utilizzate per modificare il modo in cui sono gestiti i record recuperati tramite `fetch` e i comandi eseguibili.

## Logica condizionale

All'interno del linguaggio PL/SQL, è possibile utilizzare i comandi `if`, `else`, `elsif` e `case` per controllare il flusso dei comandi stessi all'interno della sezione dei comandi eseguibili. Nel listato seguente sono riportate le sintassi delle clausole `if`:

```

if  <qualche condizione>
  then <qualche comando>
elsif <qualche condizione>
  then <qualche comando>
else <qualche comando>
end if;

```

Le condizioni if possono essere annidate le une entro le altre, come nel listato seguente:

```

if  <qualche condizione>
  then
    if <qualche condizione>
      then <qualche comando>
    end if;
  else <qualche comando>
end if;

```

Annidando le condizioni if, è possibile sviluppare rapidamente complessi flussi logici all'interno della sezione dei comandi eseguibili. Quando si annidano le condizioni if, è opportuno assicurarsi di non rendere il controllo del flusso più complesso di quanto sia necessario. Conviene verificare sempre se le condizioni logiche possono essere combinate in successioni più semplici.

L'esempio dell'area del cerchio utilizzato nel paragrafo precedente viene ora modificato in modo da contenere logica condizionale. Nel listato seguente la sezione dei comandi eseguibili del blocco PL/SQL è stata modificata in modo da contenere i comandi if e then:

```

declare
  pi      constant NUMBER(9,7) := 3.1415927;
  area   NUMBER(14,2);
  cursore rag_cursore is
    select * from VALORI_RAGGIO;
  val_rag rag_cursore%ROWTYPE;
begin
  open rag_cursore;
  fetch rag_cursore into val_rag;
  area := pi*power(val_rag.raggio,2);
  if area >30
  then
    insert into AREE values (val_rag.raggio, area);
  end if;
  close rag_cursore;
end;

```

In questo esempio, viene utilizzata una condizione if per controllare il flusso di esecuzione all'interno della sezione dei comandi eseguibili del blocco PL/SQL. I comandi di controllo del flusso sono mostrati nel listato seguente:

```

if  area >30
then
  insert into AREE values (val_rag.raggio, area);
end if;

```

In questo esempio, i comandi di controllo del flusso iniziano una volta determinato il valore della variabile “*area*”. Se il valore è maggiore di 30, un record verrà inserito nella tabella AREE. Se invece il valore è minore di 30, non verrà inserito alcun record. Questo tipo di controllo del

flusso può essere utilizzato per stabilire quale tra le varie istruzioni SQL dev'essere eseguita, in base alle condizioni specificate nelle condizioni if.

Nel listato seguente, è riportata la sintassi per il controllo di flusso che implica i comandi SQL:

```
if area>30
  then <qualche comando>
elsif area<10
  then <qualche comando>
else <qualche comando>
end if;
```

## Cicli

È possibile utilizzare i cicli per elaborare più record all'interno di un unico blocco PL/SQL. PL/SQL supporta tre tipi di cicli:

---

Cicli semplici	Tutti i cicli che continuano a ripetersi fino al raggiungimento di un'istruzione exit o exit when.
Cicli FOR	Cicli che si ripetono un determinato numero di volte.
Cicli WHILE	Cicli che si ripetono mentre una condizione è soddisfatta.

---

Nei paragrafi che seguono sono riportati esempi per ciascun tipo di ciclo che utilizzeranno come punto di partenza i blocchi PL/SQL riportati in precedenza nel capitolo. I cicli possono elaborare più record a partire da un cursore.

### Cicli semplici

Nel listato seguente, viene utilizzato un ciclo semplice per generare più righe nella tabella AREE. Il ciclo è avviato dalla parola chiave loop, mentre la clausola exit when stabilisce quando uscire dal ciclo. Una clausola end loop segnala la fine del ciclo.

```
declare
  pi    constant NUMBER(9,7) := 3.1415927;
  raggio INTEGER(5);
  area  NUMBER(14,2);
begin
  raggio := 3;
loop
  area := pi*power(raggio,2);
  insert into AREE values (raggio, area);
  raggio := raggio+1;
  exit when area >100;
end loop;
end;
```

La sezione loop dell'esempio stabilisce il controllo del flusso per i comandi nella sezione dei comandi eseguibili del blocco PL/SQL. I passaggi all'interno del ciclo sono descritti nella seguente versione commentata dei comandi loop:

```
loop
  /* Calcola l'area in base al valore del raggio.      */
  area := pi*power(raggio,2);
```

```

/* Inserisce i valori correnti nella tabella AREE. */
insert into AREE values (raggio, area);
/* Incrementa di 1 il valore del raggio. */
raggio := raggio+1;
/* Valuta l'ultima area calcolata. Se il valore      */
/* supera 100, esce. Altrimenti ripete           */
/* il ciclo con il nuovo valore del raggio.      */
exit when area >100;
/* Segnala la fine del ciclo.                      */
end loop;

```

Il ciclo dovrebbe generare più righe nella tabella AREE. Il primo record sarà quello generato da un valore di Raggio pari a 3. Quando il valore *area* supera 100, nella tabella AREE non verranno inseriti altri record.

Di seguito è riportato un output di esempio successivo all'esecuzione del blocco PL/SQL:

```

select *
from AREE
order by Raggio;

RAGGIO      AREA
-----
3          28.27
4          50.27
5          78.54
6          113.1

```

Dato che il valore di *area* per un valore di Raggio pari a 6 supera 100, non vengono elaborati ulteriori valori di Raggio e il blocco PL/SQL viene completato.

### Cicli semplici a cursore

Gli attributi di un cursore, come la presenza, o meno di righe da estrarre, possono essere utilizzati come criteri per uscire da un ciclo. Nel prossimo esempio, un cursore viene eseguito fino a quando la query non restituisce più righe. Per stabilire lo stato di un cursore, occorre verificarne gli attributi. I cursori hanno quattro attributi che si possono utilizzare nei programmi:

---

%FOUND	Il cursore può trasmettere un record.
%NOTFOUND	Il cursore non può trasmettere altri record.
%ISOPEN	Il cursore è stato aperto.
%ROWCOUNT	Numero di righe trasmesse dal cursore sino a questo momento.

---

Gli attributi del cursore %FOUND, %NOTFOUND e %ISOPEN sono valori booleani e vengono impostati a TRUE o FALSE. Poiché si tratta di attributi booleani, è possibile valutarne le impostazioni senza associarli esplicitamente a valori TRUE o FALSE. Per esempio, il comando seguente causa un'uscita quando *rag\_cursore* %NOTFOUND è TRUE:

```
exit when rag_cursore%NOTFOUND;
```

Nel listato seguente viene utilizzato un ciclo semplice per elaborare più righe di un cursore:

```

declare
  pi      constant NUMBER(9,7) := 3.1415927;
  area   NUMBER(14,2);
  cursole rag_cursole is
    select * from VALORI_RAGGIO;
  val_rag rag_cursole%ROWTYPE;
begin
  open rag_cursole;
  loop
    fetch rag_cursole into val_rag;
    exit when rag_cursole%NOTFOUND;
    area := pi*power(val_rag.raggio,2);
    insert into AREE values (val_rag.raggio, area);
  end loop;
  close rag_cursole;
end;

```

La sezione loop del blocco PL/SQL usa come input valori tratti dalla tabella VALORI\_RAGGIO. Invece di basare i criteri di uscita sul valore di Area, viene controllato l'attributo del cursore %NOTFOUND. Se nel cursore non vengono trovate altre righe, l'attributo %NOTFOUND diventerà TRUE e ciò causerà l'uscita dal ciclo. Di seguito è riportata la versione commentata del ciclo:

```

loop
  /* Nel ciclo, preleva un record.          */
  fetch rag_cursole into val_rag;
  /* Se il tentativo di prelevamento non trova */
  /* altri record nel cursore, esce dal ciclo. */
  exit when rag_cursole%NOTFOUND;
  /* Se il tentativo restituisce un record,      */
  /* elabora il valore del raggio e inserisce   */
  /* un record nella tabella AREE.               */
  area := pi*power(val_rag.raggio,2);
  insert into AREE values (val_rag.raggio, area);
  /* Segnala la fine del ciclo.                  */
end loop;

```

Quando si esegue il blocco PL/SQL precedente, tutti i record della tabella VALORI\_RAGGIO vengono elaborati dal ciclo. Finora la tabella VALORI\_RAGGIO conteneva solo un record: un valore di Raggio pari a 3. Prima di eseguire il blocco PL/SQL per questa sezione, si aggiungano due nuovi valori di Raggio alla tabella VALORI\_RAGGIO: 4 e 10.

Di seguito viene illustrata l'aggiunta dei nuovi record alla tabella VALORI\_RAGGIO:

```

insert into VALORI_RAGGIO values (4);
insert into VALORI_RAGGIO values (10);
commit;

```

```

select *
  from VALORI_RAGGIO
 order by Raggio;

RAGGIO
-----
 3
 4
10

```

Una volta aggiunti i nuovi record alla tabella VALORI\_RAGGIO, si esegua il blocco PL/SQL descritto in precedenza in questo paragrafo. Nel listato seguente è riportato l'output del blocco PL/SQL:

```
select *
  from AREE
 order by Raggio;
```

RAGGIO	AREA
3	28.27
4	50.27
10	314.16

La query sulla tabella AREE mostra come ogni record della tabella VALORI\_RAGGIO sia stato trasmesso dal cursore ed elaborato. Una volta esauriti i record da elaborare nel cursore, si esce dal ciclo e il blocco PL/SQL viene completato.

## Cicli FOR

Nei cicli semplici, il ciclo viene eseguito finché non viene soddisfatta una condizione `exit`. In un ciclo FOR, il ciclo viene eseguito per un determinato numero di volte. Di seguito è riportato un esempio di ciclo FOR. L'inizio del ciclo FOR è indicato dalla parola chiave `for`, seguita dai criteri utilizzati per stabilire quando l'elaborazione è terminata e quando si può uscire dal ciclo. Dato che il numero di esecuzioni di un ciclo viene impostato all'avvio del medesimo, all'interno del ciclo non è necessario un comando `exit`.

Nell'esempio seguente, le aree dei cerchi vengono calcolate basandosi su valori di Raggio compresi tra 1 e 7 (inclusi).

```
declare
  pi      constant NUMBER(9,7) := 3.1415927;
  raggio INTEGER(5);
  area    NUMBER(14,2);
begin
  for raggio in 1..7 loop
    area := pi*power(raggio,2);
    insert into AREE values (raggio, area);
  end loop;
end
```

I passaggi richiesti per l'elaborazione del ciclo sono visualizzati nel seguente listato commentato:

```
/* Specifica i criteri per il numero di esecuzioni */
/* del ciclo. */
for raggio in 1..7 loop
  /* Calcola l'area utilizzando il valore corrente */
  /* di Raggio. */
  area := pi*power(raggio,2);
  /* Inserisce i valori di area e raggio */
  /* nella tabella AREE. */
  insert into AREE values (raggio, area);
  /* Segnala la fine del ciclo. */
end loop;
```

Si osservi l'assenza di una linea che indica:

```
raggio := raggio+1;
```

nel ciclo FOR. Dato che la specifica del ciclo indica:

```
for raggio in 1..7 loop
```

i valori di Raggio sono già specificati. Per ciascun valore, vengono eseguiti tutti i comandi all'interno del ciclo (questi comandi possono comprendere altra logica condizionale, come le condizioni if). Una volta che il ciclo ha completato l'elaborazione di un valore di Raggio, vengono verificati i limiti nella clausola for e si passa, a seconda dei casi, a elaborare il successivo valore di Raggio o a completare l'esecuzione del ciclo.

Nel listato seguente è mostrato un output di esempio ricavato dall'esecuzione del ciclo FOR:

```
select *
  from AREE
 order by Raggio;
```

RAGGIO	AREA
1	3.14
2	12.57
3	28.27
4	50.27
5	78.54
6	113.1
7	153.94

7 rows selected.

### Cicli FOR a cursore

Nei cicli FOR, le iterazioni vengono eseguite per un numero specifico di volte. In un ciclo FOR a cursore, i risultati di una query servono per stabilire in modo dinamico il numero di esecuzioni del ciclo. In questi cicli l'apertura, la trasmissione e la chiusura dei cursori sono eseguite implicitamente; quindi, non è necessario specificare queste azioni in modo esplicito.

Di seguito è riportato un ciclo FOR a cursore che esegue una query sulla tabella VALORI\_RAGGIO e inserisce dei record nella tabella AREE:

```
declare
  pi      constant NUMBER(9,7) := 3.1415927;
  area   NUMBER(14,2);
  cursore rag_cursore is
    select * from VALORI_RAGGIO;
begin
  for val_rag in rag_cursore
  loop
    area := pi*power(val_rag.raggio,2);
    insert into AREE values (val_rag.raggio, area);
  end loop;
end
```

In un ciclo FOR a cursore non compare un comando open o fetch. Il comando:

```
for val_rag in rag_cursore
```

apre implicitamente il cursore “*rag\_cursore*” e trasmette un valore nella variabile “*val\_rag*”. Quando nel cursore non ci sono più record, si esce dal ciclo e il cursore viene chiuso. In un ciclo FOR a cursore, il comando *close* non è necessario. Si osservi che la variabile *val\_rag* non è dichiarata esplicitamente nel blocco.

Nel listato seguente è visibile la porzione loop del blocco PL/SQL, con commenti che descrivono il flusso del controllo. Il ciclo è controllato dall'esistenza di un record trasmissibile nel cursore “*rag\_cursore*”. Non è necessario verificare l'attributo %NOTFOUND del cursore, perché a ciò provvede automaticamente il ciclo FOR a cursore.

```
/* Se un record può essere prelevato dal cursore, */
/* lo memorizza nella variabile val_rag. Se          */
/* non ci sono righe da prelevare, salta il ciclo. */
for val_rag in rag_cursore
/* Iniziano i comandi di ciclo.                      */
loop
/* Calcola l'area in base al valore del raggio   */
/* e inserisce un record nella tabella AREE.      */
area := pi*power(val_rag.raggio,2);
insert into AREE values (val_rag.raggio, area);
/* Segnala la fine dei comandi di ciclo.           */
end loop;
```

Nel listato seguente è riportato un output di esempio; per questo caso la tabella VALORI\_RAGGIO dispone di tre record, con valori di Raggio pari a 3, 4 e 10.

```
select *
  from VALORI_RAGGIO
order by Raggio;
-----  
RAGGIO  
-----  
3  
4  
10
```

L'esecuzione del blocco PL/SQL con il ciclo FOR a cursore genera i record seguenti nella tabella AREE:

```
select *
  from AREE
order by Raggio;
-----  
RAGGIO      AREA  
-----  
3          28.27  
4          50.27  
10         314.16
```

## Cicli WHILE

In un ciclo WHILE, il ciclo viene elaborato finché non viene soddisfatta una condizione di uscita. Invece di specificare all'interno del ciclo la condizione di uscita tramite un comando *exit*, questa condizione può essere specificata nel comando *while* che avvia il ciclo.

Nell'esempio seguente, viene creato un ciclo WHILE che consente l'elaborazione di più valori di Raggio. Se il valore corrente della variabile Raggio soddisfa la condizione while nella

specifica del ciclo, i comandi del ciclo vengono elaborati. Quando un valore di Raggio non soddisfa la condizione while nella specifica del ciclo, l'esecuzione di quest'ultimo termina:

```
declare
  pi    constant NUMBER(9,7) := 3.1415927;
  raggio INTEGER(5);
  area   NUMBER(14,2);
begin
  raggio := 3;
  while raggio<=7
  loop
    area := pi*power(raggio,2);
    insert into AREE values (raggio, area);
    raggio := raggio+1;
  end loop;
end;
```

Il ciclo WHILE è simile nella struttura a un ciclo semplice, in quanto termina basandosi sul valore di una variabile. Di seguito sono riportati i passaggi implicati nel ciclo e i relativi commenti:

```
/* Imposta un valore iniziale per la variabile Raggio. */
raggio := 3;
/* Stabilisce i criteri per terminare il ciclo. */
/* Se la condizione viene soddisfatta, esegue */
/* i comandi all'interno del ciclo. Se la condizione */
/* non è soddisfatta, termina il ciclo. */
while raggio<=7
/* Inizia i comandi da eseguire. */
loop
/* Calcola l'area in base al valore corrente */
/* di Raggio e inserisce un record */
/* nella tabella AREE. */
area := pi*power(val_rag.raggio,2);
insert into AREE values (val_rag.raggio, area);
/* Imposta un nuovo valore per la variabile Raggio. */
/* Questo nuovo valore viene confrontato con i criteri */
/* di uscita e i comandi del ciclo vengono eseguiti */
/* per il nuovo valore di Raggio. */
/* oppure il ciclo termina. */
raggio := raggio+1;
/* Segnala la fine dei comandi all'interno del ciclo. */
end loop;
```

Al momento dell'esecuzione, il blocco PL/SQL dell'esempio precedente inserisce dei record nella tabella AREE. Di seguito è riportato l'output del blocco PL/SQL:

```
select *
  from AREE
 order by Raggio;
```

RAGGIO	AREA
3	28.27
4	50.27

```

5      78.54
6      113.1
7      153.94

```

A causa del valore assegnato alla variabile “Raggio” prima del ciclo, questo deve forzatamente essere eseguito almeno una volta. Sarebbe opportuno controllare che gli assegnamenti delle variabili soddisfino le condizioni utilizzate per limitare le esecuzioni del ciclo.

## Istruzioni CASE

A partire da Oracle9i è possibile utilizzare le istruzioni case per controllare la logica di selezione nei blocchi PL/SQL. Per esempio, queste istruzioni possono servire per assegnare dei valori ponendo delle condizioni oppure per trasformare valori prima di inserirli. Nell'esempio seguente, le espressioni case usano i valori di Raggio per decidere quali righe inserire nella tabella AREE:

```

declare
    pi      constant NUMBER(9,7) := 3.1415927;
    area    NUMBER(14,2);
    cursore rag_cursore is
        select * from VALORI_RAGGIO;
    val_rag rag_cursore%ROWTYPE;
begin
    open rag_cursore;
    loop
        fetch rag_cursore into val_rag;
        exit when ra_cursore%NOTFOUND;
        area := pi*power(val_rag.raggio,2);
        case
            when val_rag.Raggio = 3
            then
                insert into AREE values (val_rag.raggio, area);
            when val_rag.Raggio = 4
            then
                insert into AREE values (val_rag.raggio, area);
            when val_rag.Raggio = 10
            then
                insert into AREE values (0, 0);
            else raise CASE_NOT_FOUND;
        end case;
    end loop;
    close rag_cursore;
end;

```

Questo blocco crea l'output seguente. La clausola case ha indotto l'inserimento del valore Raggio pari a 10 nei valori di Raggio e Area pari a 0:

```
select * from AREE;
```

RAGGIO	AREA
3	28.27
4	50.27
0	0

La parola chiave **case** dà inizio alla clausola:

```
case
when val_rag.Raggio = 3
then
    insert into AREE values (val_rag.raggio, area);
```

Le clausole **when** vengono valutate in sequenza. La parola chiave **else** all'interno della clausola **case** funziona in modo analogo alla parola chiave **else** in una clausola **if-then**. Se si tralascia la parola chiave **else**, PL/SQL aggiunge la seguente clausola **else** implicita:

```
else raise CASE_NOT_FOUND;
```

La clausola **end case** termina la clausola **case**. La clausola **case** viene utilizzata spesso per convertire elenchi di valori nelle rispettive descrizioni, come nel caso dei nomi di categorie della libreria:

```
case NomeCategoria
when 'NARRADULTI' then 'Narrativa adulti'
when 'SAGGIADULTI' then 'Saggi adulti'
when 'CONSADULTI' then 'Consultazione adulti'
when 'NARRRAGAZZI' then 'Narrativa ragazzi'
when 'SAGGIRAGAZZI' then 'Saggi ragazzi'
when 'ILLUSRAGAZZI' then 'Illustrati ragazzi'
else NomeCategoria
end;
```

Per esempi relativi agli impieghi complessi di **case**, si rimanda al Capitolo 17.

## 27.4 La sezione di gestione delle eccezioni

Quando si incontrano eccezioni (errori) definite dall'utente o relative al sistema, il controllo del blocco PL/SQL passa alla sezione di gestione delle eccezioni. All'interno di questa sezione, una clausola **when** viene utilizzata per valutare quale sia l'eccezione da "sollevare", ovvero eseguire.

Se all'interno della sezione dei comandi eseguibili del blocco PL/SQL viene sollevata un'eccezione, il flusso dei comandi abbandona immediatamente tale sezione e cerca nella sezione di gestione delle eccezioni un'eccezione che coincida con l'errore incontrato. Il linguaggio PL/SQL mette a disposizione una serie di eccezioni definite dal sistema e consente di aggiungere eccezioni personalizzate. Esempi di eccezioni definite dall'utente sono mostrati nei Capitoli 28 e 29.

La sezione di gestione delle eccezioni inizia sempre con la parola chiave **exception** e precede il comando **end** che termina la sezione dei comandi eseguibili del blocco PL/SQL. Nel listato seguente, è mostrata la posizione della sezione di gestione delle eccezioni all'interno del blocco PL/SQL:

```
declare
    <sezione dichiarazioni>
begin
    <comandi eseguibili>
exception
    <gestione delle eccezioni>
end;
```

La sezione di gestione delle eccezioni di un blocco PL/SQL è facoltativa. Nessuno dei blocchi PL/SQL presentati in precedenza in questo capitolo comprendeva una sezione di questo tipo. Tuttavia, gli esempi precedenti si basavano su un gruppo molto piccolo di valori di input noti, con elaborazione molto ridotta.

Nel listato seguente, è riportato il semplice ciclo per il calcolo dell'area di un cerchio, con due modifiche (in grassetto). Nella sezione delle dichiarazioni viene dichiarata una nuova variabile di nome “*qualche\_variabile*”, mentre nella sezione dei comandi eseguibili viene creato un calcolo per stabilire il valore della variabile.

```
declare
    pi      constant NUMBER(9,7) := 3.1415927;
    raggio INTEGER(5);
    area   NUMBER(14,2);
    qualche_variabile  NUMBER(14,2);
begin
    raggio := 3;
    loop
        qualche_variabile := 1/(raggio-4);
        area := pi*power(raggio,2);
        insert into AREE values (raggio, area);
        raggio := raggio+1;
        exit when area >100;
    end loop;
end;
```

Dato che il calcolo di “*qualche\_variabile*” richiede una divisione, è possibile incontrare una situazione in cui il calcolo cerca di dividere per zero, provocando un errore. La prima volta che si esegue il ciclo, viene elaborata la variabile Raggio (con un valore iniziale pari a 3) e viene inserito un record nella tabella AREE. Alla seconda esecuzione del ciclo, la variabile “Raggio” ha valore pari a 4 e il calcolo di “*qualche\_variabile*” incontra un errore.

```
declare
*
ERROR at line 1:
ORA-01476: divisor is equal to zero
ORA-06512: at line 9
```

A causa dell'errore, la prima riga inserita in AREE subisce un rollback e il blocco PL/SQL termina.

L'elaborazione della condizione di errore può essere modificata aggiungendo una sezione di gestione delle eccezioni al blocco PL/SQL, come mostrato nel listato seguente:

```
declare
    pi      constant NUMBER(9,7) := 3.1415927;
    raggio INTEGER(5);
    area   NUMBER(14,2);
    variabile  NUMBER(14,2);
begin
    raggio := 3;
    loop
        variabile := 1/(raggio-4);
        area := pi*power(raggio,2);
        insert into AREE values (raggio, area);
        raggio := raggio+1;
    end loop;
end;
```

```

    exit when area >100;
end loop;
exception
when ZERO_DIVIDE
    then insert into AREE values (0,0);
end;

```

Ecco la sezione di gestione delle eccezioni del blocco PL/SQL:

```

exception
when ZERO_DIVIDE
    then insert into AREAE values (0,0);

```

Quando si verifica un errore, il blocco PL/SQL ricerca le eccezioni definite nella sezione di gestione delle eccezioni. In questo caso, il blocco trova l'eccezione ZERO\_DIVIDE, una delle eccezioni definite dal sistema disponibili nel linguaggio PL/SQL. Oltre alle eccezioni definite dal sistema e a quelle definite dall'utente, è possibile utilizzare la clausola `when others` per indirizzare tutte le eccezioni non definite all'interno della sezione di gestione delle eccezioni. All'interno della sezione di gestione delle eccezioni, viene eseguito il comando relativo all'eccezione corrispondente e nella tabella AREE viene inserita una riga. Di seguito è riportato l'output del blocco PL/SQL:

```

select *
from AREA;

```

RAGGIO	AREA
3	28.27
0	0

L'output mostra che il primo valore di “Raggio” (3) è stato elaborato e che l'errore si è verificato durante la seconda esecuzione del ciclo.

**NOTA** *Una volta incontrata un'eccezione, non è possibile tornare al normale flusso dell'elaborazione dei comandi all'interno della sezione dei comandi eseguibili. Se si deve mantenere il controllo all'interno della sezione dei comandi eseguibili, occorre utilizzare delle condizioni if per verificare la possibile presenza di eccezioni prima che queste vengano incontrate dal programma, oppure creare un blocco annidato dotato di una propria sezione locale di gestione delle eccezioni.*

Alla voce “Eccezioni” del Capitolo 42 sono elencate tutte le eccezioni definite dal sistema. Nei Capitoli 28 e 29 sono riportati esempi di eccezioni definite dall'utente.

## Capitolo 28

# I trigger

- 28.1 **Privilegi di sistema richiesti**
- 28.2 **Privilegi di tabella richiesti**
- 28.3 **Tipi di trigger**
- 28.4 **Sintassi dei trigger**
- 28.5 **Attivazione e disattivazione dei trigger**
- 28.6 **Sostituzione di trigger**
- 28.7 **Eliminazione di trigger**

**U**n trigger definisce un’azione che il database deve intraprendere quando si verifica un determinato evento correlato al database stesso. I trigger possono essere utilizzati per migliorare l’integrità referenziale dichiarativa, per imporre regole complesse legate all’attività o per effettuare revisioni sulle modifiche ai dati. Il codice interno a un trigger, detto *corpo del trigger*, è costituito da blocchi PL/SQL (consultare il Capitolo 27).

All’utente l’esecuzione dei trigger risulta trasparente. I trigger vengono eseguiti dal database quando tipi specifici di comandi per la manipolazione dei dati vengono eseguiti su tabelle ben determinate. Tra questi comandi troviamo insert, update e delete. Gli aggiornamenti di colonne specifiche possono essere utilizzati anche come eventi che attivano il trigger. A partire da Oracle8i, gli eventi che attivano i trigger possono includere anche i comandi DDL e gli eventi del database (come chiusure e connessioni).

Grazie alla loro flessibilità, i trigger possono aggiungere integrità referenziale, ma non possono essere utilizzati in sostituzione di questa. Quando in un’applicazione si impongono regole legate all’attività, per prima cosa è necessario affidarsi all’integrità referenziale dichiarativa disponibile in Oracle; quindi si potranno utilizzare trigger per imporre regole che non possono essere codificate attraverso l’integrità referenziale.

## 28.1 Privilegi di sistema richiesti

Per creare un trigger su una tabella, si deve essere in grado di modificare la tabella stessa. Pertanto, è necessario possedere la tabella, disporre del privilegio ALTER per la tabella o del privilegio di sistema ALTER ANY TABLE. Inoltre occorre avere il privilegio di sistema CREATE TRIGGER. Per creare trigger nell’account di un altro utente (definito anche *schema*), si deve disporre del privilegio di sistema CREATE ANY TRIGGER. Il privilegio di sistema CREATE TRIGGER fa parte del ruolo RESOURCE previsto da Oracle.

Per *modificare* un trigger, è necessario esserne il proprietario o possedere il privilegio di sistema ALTER ANY TRIGGER. È altresì possibile modificare i trigger intervenendo sulle tabelle su cui essi si basano. Ciò richiede la disponibilità del privilegio ALTER per la tabella che si desidera modificare o del privilegio di sistema ALTER ANY TABLE. Per informazioni sulla modifica dei trigger, si rimanda al paragrafo “Attivazione e disattivazione dei trigger” più avanti in questo capitolo.

Per creare un trigger su un evento avvenuto a livello di database, è necessario il privilegio di sistema ADMINISTER DATABASE TRIGGER.

## 28.2 Privilegi di tabella richiesti

I trigger possono fare riferimento a tabelle differenti da quella che attiva l'evento del trigger. Per esempio, se si usano dei trigger per l'audit delle modifiche ai dati nella tabella BIBLIOTECA, è possibile inserire un record in una tabella differente (come BIBLIOTECA\_AUDIT) ogni volta che un record della tabella BIBLIOTECA viene modificato. A tal fine occorre disporre dei privilegi necessari a eseguire inserimenti nella tabella BIBLIOTECA\_AUDIT (per effettuare la transazione attivata con trigger).

**NOTA** *I privilegi necessari per le transazioni attivate con trigger non possono derivare da ruoli. Tali privilegi devono essere concessi direttamente al creatore del trigger.*

## 28.3 Tipi di trigger

Il tipo di un trigger è definito dal tipo di transazione che lo attiva e dal livello al quale il trigger viene eseguito. Nei paragrafi seguenti sono riportate le descrizioni di queste classificazioni, insieme alle limitazioni più importanti.

### Trigger a livello di riga

*I trigger a livello di riga* vengono eseguiti una volta sola per ogni riga su cui agisce un'istruzione DML. Per l'esempio di audit della tabella BIBLIOTECA descritto in precedenza, ogni riga modificata in questa tabella può essere elaborata dal trigger. I trigger a livello di riga costituiscono il tipo più comune; vengono utilizzati spesso in applicazioni di audit dei dati. Inoltre si rivelano utili per mantenere sincronizzati i dati distribuiti. Le viste materializzate che utilizzano per questo scopo dei trigger a livello di riga interni sono descritte nel Capitolo 23.

Per creare trigger a livello di riga occorre utilizzare la clausola `for each row` nel comando `create trigger`. La sintassi dei trigger è mostrata nel paragrafo “Sintassi dei trigger”, più avanti in questo capitolo.

### Trigger a livello di istruzione

*I trigger a livello di istruzione* vengono eseguiti una sola volta per ogni istruzione DML. Per esempio, se una singola istruzione `INSERT` avesse inserito 500 righe nella tabella BIBLIOTECA, un trigger a livello di istruzione su tale tabella verrebbe eseguito una sola volta. Pertanto i trigger a livello di istruzione non vengono utilizzati di frequente per attività correlate ai dati. In genere questi trigger vengono impiegati per imporre ulteriori misure di sicurezza sui tipi di azioni che possono essere eseguite su una tabella.

I trigger a livello di istruzione sono il tipo predefinito creato dal comando `create trigger`. La sintassi dei trigger è mostrata nel paragrafo “Sintassi dei trigger”, più avanti in questo capitolo.

## Trigger BEFORE e AFTER

Dato che i trigger vengono eseguiti da eventi, è possibile impostarli in modo che si verifichino immediatamente prima o dopo tali eventi. Dal momento che tra gli eventi che eseguono i trigger sono incluse anche le istruzioni DML di database, i trigger possono essere eseguiti immediatamente prima o dopo gli inserimenti, gli aggiornamenti e le cancellazioni. Per gli eventi avvenuti a livello di database, vengono applicate altre limitazioni; non è possibile attivare un evento in modo che si verifichi prima di una connessione o di un avvio.

All'interno del trigger è possibile fare riferimento ai valori vecchi e nuovi coinvolti nell'istruzione DML. L'accesso richiesto per i vecchi e i nuovi dati può determinare quale tipo di trigger sia necessario. "Vecchio" indica i dati così come erano prima dell'istruzione DML; gli aggiornamenti e le cancellazioni in genere fanno riferimento a valori vecchi. I valori "nuovi" sono i valori dei dati creati dall'istruzione DML (come le colonne in un record inserito).

Se si deve impostare il valore di una colonna in una riga inserita mediante un trigger, è necessario utilizzare un trigger BEFORE INSERT per accedere ai valori "nuovi". L'uso di un trigger AFTER INSERT non consentirebbe di impostare il valore inserito, in quanto la riga sarebbe già stata inserita nella tabella.

I trigger AFTER a livello di riga vengono utilizzati spesso in applicazioni di audit, in quanto non si attivano sino a che la riga non è stata modificata. La modifica riuscita di una riga implica il superamento dei vincoli di integrità referenziale definiti per la tabella.

## Trigger INSTEAD OF

È possibile utilizzare trigger INSTEAD OF per indicare a Oracle che cosa fare *invece di* eseguire le azioni che hanno invocato il trigger. Per esempio, è possibile utilizzare un trigger INSTEAD OF su una vista per reindirizzare gli inserimenti in una tabella o per aggiornare più tabelle che facciano parte di una vista. I trigger INSTEAD OF possono essere utilizzati sia su viste oggetto (consultare il Capitolo 30), sia su viste relazionali.

Per esempio, se una vista comporta il join di due tabelle, la possibilità di utilizzare il comando update su record della vista è limitata. Tuttavia, con un trigger INSTEAD OF è possibile specificare a Oracle come eseguire aggiornamenti, cancellazioni e inserimenti di record nelle tabelle base della vista quando un utente cerca di modificare i valori attraverso la vista stessa. Il codice contenuto nel trigger INSTEAD OF viene eseguito al posto del comando insert, update o delete inserito.

In questo capitolo, si imparerà a implementare i trigger di base. I trigger INSTEAD OF, inizialmente introdotti per supportare le viste oggetto, sono descritti nel Capitolo 30.

## Trigger di schema

Esiste la possibilità di creare i trigger su operazioni eseguite a livello di schema, come create table, alter table, drop table, audit, rename, truncate e revoke. Inoltre, è possibile creare dei trigger per impedire agli utenti di eliminare le proprie tabelle. I trigger a livello di schema offrono principalmente due capacità: impedire l'esecuzione di operazioni DDL e fornire ulteriore controllo della sicurezza quando si verificano operazioni DDL.

## Trigger a livello di database

È possibile creare dei trigger che vengono attivati per eventi di database, compresi errori, collegamenti, scollegamenti, chiusure e avvii. Questo genere di trigger può servire ad automatizzare le operazioni di gestione e audit del database.

## 28.4 Sintassi dei trigger

La sintassi completa per il comando `create trigger` è riportata nel Capitolo 42. Il listato seguente contiene una versione abbreviata della sintassi del comando:

```
create [or replace] trigger [schema .] trigger
{ before | after | instead of }
{ clausola_evento_dml
| { evento_ddl [or evento_ddl ]...
| evento_database [or evento_database ]...
}
on { [schema .] schema | database }
}
[when ( condizione ) ]
{ blocco_pl/sql| istruzione_procedura_chiamata }
```

Le opzioni sintattiche disponibili dipendono dal tipo di trigger utilizzato. Per esempio, un trigger su un evento DML utilizzerà `clausola_evento_dml`, che segue questa sintassi:

```
}]...
on { [schema .] tabella | [nested table colonna_tabella_annidata of] [schema .] vista }
[clausola_riferimento] [for each row]
```

Nella progettazione di un trigger è consentita una grande flessibilità. Le parole chiave `before` e `after` indicano se il trigger dovrà essere eseguito prima o dopo la transazione che lo attiva. Con la clausola `instead of` il codice del trigger verrà eseguito al posto dell'evento che ha attivato il trigger. Le parole chiave `delete`, `insert` e `update` (l'ultima può comprendere un elenco di colonne) indicano il tipo di manipolazione dei dati che costituirà un evento in grado di attivare il trigger. Quando si fa riferimento ai valori vecchi e nuovi delle colonne, è possibile utilizzare i valori predefiniti (“old” e “new”) o servirsi della clausola `referencing` per specificare altri nomi.

Con la clausola `for each row` si ha un trigger a livello di riga, altrimenti si ha un trigger a livello di istruzione. La clausola `when` viene utilizzata per applicare ulteriori restrizioni quando si esegue il trigger. Le restrizioni imposte nella clausola `when` possono comprendere verifiche dei vecchi e nuovi valori dei dati.

Per esempio, si supponga di voler tener traccia di qualsiasi modifica al valore Classificazione della tabella BIBLIOTECA ogni volta che i valori di classificazione vengono ridotti. Per prima cosa, si creerà una tabella che memorizzerà i record di audit:

```
drop table AUDIT_BIBLIOTECA;
create table AUDIT_BIBLIOTECA
(Titolo      VARCHAR2(100),
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Vecchia_Classificazione  VARCHAR2(2),
Nuova_Classificazione   VARCHAR2(2),
Data_Audit    DATE);
```

Il seguente trigger a livello di riga BEFORE UPDATE viene eseguito solo se il valore di Classificazione viene ridotto. L'esempio seguente descrive anche l'uso della parola chiave `new`, che fa riferimento al nuovo valore della colonna, e di `old` che fa riferimento al vecchio valore della colonna.

```

create or replace trigger BIBLIOTECA_BEF_UPD_ROW
before update on BIBLIOTECA
for each row
when (nuova.Classificazione = vecchia.Classificazione)
begin
  insert into AUDIT_BIBLIOTECA
    (Titolo, Editore, NomeCategoria,
     Vecchia_Classificazione, Nuova_Classificazione, Data_Audit)
  values
    (:old.Titolo, :old.Editore, :old.NomeCategoria,
     :old.Classificazione, :new.Classificazione, Sysdate);
end;

```

La suddivisione di questo comando `create trigger` nei suoi componenti ne rende più semplice la comprensione. In primo luogo viene assegnato un nome al trigger:

```
create or replace trigger BIBLIOTECA_BEF_UPD_ROW
```

Il nome del trigger contiene il nome della tabella su cui agisce e il tipo del trigger stesso (per informazioni sulle convenzioni di denominazione, si rimanda al paragrafo “Assegnazione dei nomi ai trigger” più avanti in questo capitolo).

Questo trigger si applica alla tabella BIBLIOTECA; verrà eseguito prima che le transazioni di aggiornamento siano state confermate con un `commit` nel database:

```
before update on BIBLIOTECA
```

Vista la presenza della clausola `for each row`, il trigger verrà applicato a ciascuna riga modificata dall’istruzione `update`. Se non si utilizzata questa clausola, il trigger verrà eseguito a livello delle istruzioni.

```
for each row
```

La clausola `when` aggiunge ulteriori criteri alla condizione di attivazione del trigger. L’evento che attiva il trigger non solo dovrà essere un aggiornamento della tabella BIBLIOTECA, ma dovrà anche riflettere una riduzione del valore di `Classificazione`:

```
when (nuova.valutazione < vecchia.Classificazione)
```

Il codice PL/SQL visibile nel listato seguente rappresenta il corpo del trigger. I comandi in esso contenuti verranno eseguiti a ogni aggiornamento della tabella BIBLIOTECA che soddisfi la condizione `when`. Perché ciò accada, deve esistere la tabella AUDIT\_BIBLIOTECA e al proprietario del trigger devono essere stati concessi dei privilegi (direttamente e non tramite ruoli) su questa tabella. Questo esempio inserisce i vecchi valori dal record BIBLIOTECA nella tabella AUDIT\_BIBLIOTECA prima che il record BIBLIOTECA venga aggiornato.

```

begin
  insert into AUDIT_BIBLIOTECA
    (Titolo, Editore, NomeCategoria,
     Vecchia_Classificazione, Nuova_Classificazione, Data_Audit)
  values
    (:old.Titolo, :old.Editore, :old.NomeCategoria,
     :old.Classificazione, :new.Classificazione, Sysdate);
end;

```

**NOTA** Quando nel blocco PL/SQL si fa riferimento alle parole chiave new e old, queste sono precedute dal segno di due punti (:).

Questo è un tipico esempio di trigger di audit. L'attività di audit risulta completamente trasparente all'utente che esegue l'aggiornamento della tabella BIBLIOTECA. Tuttavia, la transazione eseguita sulla tabella BIBLIOTECA dipende dal successo dell'esecuzione del trigger.

## Combinazioni di tipi di trigger DML

I trigger per più comandi insert, update e delete su una tabella possono essere combinati in un singolo trigger, purché siano tutti dello stesso livello (di riga o di istruzione). Nell'esempio seguente viene descritto un trigger che viene eseguito ogniqualvolta avviene un inserimento o un aggiornamento. In questo esempio, diversi punti di particolare importanza sono evidenziati in grassetto:

- La porzione update del trigger viene eseguita solo quando il valore della colonna Classificazione viene aggiornato.
- All'interno del blocco PL/SQL viene utilizzata una clausola if per stabilire quale tra i due comandi ha invocato il trigger.
- In questo esempio, la colonna da modificare è specificata nella *clausola\_evento\_dml* e non nella clausola when come accadeva negli esempi precedenti.

```
drop trigger BIBLIOTECA_BEF_UPD_ROW;

create or replace trigger BIBLIOTECA_BEF_UPD_INS_ROW
before insert or update of Classificazione on BIBLIOTECA
for each row
begin
  if INSERTING then
    insert into AUDIT_BIBLIOTECA
      (Titolo, Editore, NomeCategoria,
       Nuova_Classificazione, Data_Audit)
    values
      (:new.Titolo, :new.Editore, :new.NomeCategoria,
       :new.Classificazione, Sysdate);
  else -- se non è un inserimento, si sta aggiornando Classificazione
    insert into AUDIT_BIBLIOTECA
      (Titolo, Editore, NomeCategoria,
       Vecchia_Classificazione, Nuova_Classificazione, Data_Audit)
    values
      (:old.Titolo, :old.Editore, :old.NomeCategoria,
       :old.Classificazione, :new.Classificazione, Sysdate);
  end if;
end;
```

Si osservino nuovamente i componenti del trigger. Per prima cosa, il trigger riceve un nome e viene identificato come trigger before insert e before update (del valore Classificazione), da eseguirsi per ogni riga (for each row).

```
create or replace trigger BIBLIOTECA_BEF_UPD_INS_ROW
before insert or update of Classificazione on BIBLIOTECA
for each row
```

Quindi segue il corpo del trigger, nella prima del quale, riportata di seguito, avviene il controllo del tipo di transazione tramite una clausola if. I tipi di transazioni validi sono INSERTING, DELETING e UPDATING. In questo caso, il trigger verifica se il record dev'essere inserito nella tabella BIBLIOTECA. In caso affermativo, viene eseguita la prima parte del corpo del trigger. La parte INSERTING del corpo del trigger inserisce i nuovi valori del record nella tabella AUDIT\_BIBLIOTECA.

```
begin
  if INSERTING then
    insert into AUDIT_BIBLIOTECA
      (Titolo, Editore, NomeCategoria,
       Nuova_Classificazione, Data_Audit)
    values
      (:new.Titolo, :new.Editore, :new.NomeCategoria,
       :new.Classificazione, Sysdate);
```

Possono quindi essere verificati altri tipi di transazioni. In questo esempio, dal momento che il trigger è stato eseguito, la transazione dev'essere un inserimento o un aggiornamento della colonna Classificazione. Dato che la clausola if nella prima metà del corpo del trigger verifica gli inserimenti, e il trigger viene eseguito solo per inserimenti e aggiornamenti, le uniche condizioni in grado di eseguire la seconda metà del corpo del trigger sono gli aggiornamenti di Classificazione. Di conseguenza, non sono necessarie altre clausole if per determinare il tipo di evento DML. Questa parte del corpo del trigger è identica a quella precedente: prima di essere aggiornati, i vecchi valori nella riga vengono scritti nella tabella AUDIT\_BIBLIOTECA.

```
else -- se non è un inserimento, si sta aggiornando Classificazione
  insert into AUDIT_BIBLIOTECA
    (Titolo, Editore, NomeCategoria,
     Vecchia_Classificazione, Nuova_Classificazione, Data_Audit)
  values
    (:old.Titolo, :old.Editore, :old.NomeCategoria,
     :old.Classificazione, :new.Classificazione, Sysdate);
```

Questa combinazione di tipi differenti di trigger può agevolare la coordinazione dello sviluppo di trigger tra più sviluppatori, in quanto consolida tutti gli eventi di database che dipendono da un'unica tabella.

## Impostazione di valori inseriti

I trigger possono essere utilizzati per impostare valori di colonna durante gli inserimenti e gli aggiornamenti. Negli esempi proposti in precedenza nel capitolo, il valore Data\_Audit di AUDIT\_BIBLIOTECA è stato impostato al risultato della funzione SYSDATE. Anche gli aggiornamenti possono servire a supportare esigenze di applicazioni differenti, come la memorizzazione di una versione di un valore scritta in maiuscolo insieme alla versione inserita dagli utenti in cui maiuscole e minuscole si mescolano. In questo caso, la tabella potrebbe essere stata modificata parzialmente in modo da includere una colonna per i dati derivati. La memorizzazione di questi dati in maiuscolo (per questo esempio, nella colonna PersonaMaiusc) consente agli utenti di visualizzare i dati nel loro formato naturale, pur utilizzando la colonna maiuscola durante le query.

Dato che la versione maiuscola del valore corrisponde ai dati derivati, si potrebbe perdere la sincronizzazione con la colonna inserita dall'utente. A meno che l'applicazione non fornisca un

valore per la versione maiuscola durante gli inserimenti, quando si inserisce una nuova riga questo valore della colonna sarà NULL.

Per evitare questo problema di sincronismo, è possibile utilizzare un trigger di database. Si inserisca un trigger BEFORE INSERT e uno BEFORE UPDATE nella tabella; questi due trigger agiranno a livello di riga. Come mostrato nel listato seguente, con questo metodo è possibile impostare un nuovo valore per NomeMaiusc ogni volta che il valore della colonna Nome cambia all'interno della tabella BIBLIOTECA\_PRESTITO:

```
alter table BIBLIOTECA_PRESTITO add (NomeMaiusc VARCHAR2(25));

create or replace trigger BIBLIOTECA_PRESTITO_BUI_ROW
before insert or update of Nome on BIBLIOTECA_PRESTITO
for each row
begin
    :new.NomeMaiusc := UPPER(:new.Nome);
end;
```

In questo esempio, il corpo del trigger determina il valore di NomeMaiusc utilizzando la funzione UPPER sulla colonna Nome. Questo trigger verrà eseguito ogni volta che si inserisce una riga nella tabella BIBLIOTECA\_PRESTITO e ogni volta che si aggiorna la colonna Nome. Le colonne Nome e NomeMiausc non perderanno la loro sincronizzazione.

## Conservazione di dati duplicati

Il metodo per l'impostazione di valori tramite trigger descritto nel paragrafo precedente può essere combinato con i metodi di accesso ai dati remoti analizzati nel Capitolo 22. Come accade con le viste materializzate, è possibile replicare tutte o solo alcune righe di una tabella.

Per esempio, si potrebbe creare e gestire una copia del log di audit dell'applicazione. In questo modo, ci si protegge dall'eliminazione, da parte di un'unica applicazione, dei record di log di audit creati da più applicazioni. La duplicazione dei log di audit viene utilizzata spesso nel monitoraggio di sicurezza.

Si consideri la tabella AUDIT\_BIBLIOTECA proposta negli esempi di questo capitolo. Si potrebbe creare una seconda tabella, DUP\_AUDIT\_BIBLIOTECA, eventualmente in un database remoto. Per questo esempio, si supponga di poter utilizzare un database link, di nome AUDIT\_LINK, per connettere l'utente al database in cui risiede DUP\_AUDIT\_BIBLIOTECA (per ulteriori informazioni sui database link, si rimanda al Capitolo 22).

```
drop table DUP_AUDIT_BIBLIOTECA;
create table DUP_AUDIT_BIBLIOTECA
(Titolo      VARCHAR2(100),
Editore     VARCHAR2(20),
NomeCategoria VARCHAR2(20),
Vecchia_Classificazione  VARCHAR2(2),
Nuova_Classificazione   VARCHAR2(2),
Data_Audit    DATE);
```

Per automatizzare il popolamento della tabella DUP\_AUDIT\_BIBLIOTECA, si potrebbe collocare questo trigger sulla tabella AUDIT\_BIBLIOTECA:

```
create or replace trigger BIBLIOTECA_AUDIT_AFT_INS_ROW
after insert on AUDIT_BIBLIOTECA
for each row
```

---

```

begin
  insert into DUP_AUDIT_BIBLIOTECA@AUDIT_LINK
    (Titolo, Editore, NomeCategoria,
     Nuova_Classificazione, Data_Audit)
   values (:new.Titolo, :new.Editore, :new.NomeCategoria,
           :new.Nuova_Classificazione, :new.Data_Audit);
end;

```

Come mostra l'intestazione, questo trigger viene eseguito per ogni riga inserita nella tabella AUDIT\_BIBLIOTECA. Esso inserisce un singolo record nella tabella DUP\_AUDIT\_BIBLIOTECA, nel database definito dal database link AUDIT\_LINK. AUDIT\_LINK può puntare a un database che si trova su un server remoto. Per i requisiti di replica asincrona, si prenda in considerazione l'eventualità di utilizzare le viste materializzate (consultare il Capitolo 23).

È possibile vedere le azioni di questi trigger inserendo una riga nella tabella BIBLIOTECA:

```

insert into BIBLIOTECA
(Titolo, Editore, NomeCategoria, Classificazione) values
('HARRY POTTER AND THE CHAMBER OF SECRETS',
'SCHOLASTIC', 'NARRAGAZZI', '4');

```

1 row created.

```
select Titolo from AUDIT_BIBLIOTECA;
```

TITOLO

---

HARRY POTTER AND THE CHAMBER OF SECRETS

```
select Titolo from DUP_AUDIT_BIBLIOTECA;
```

TITOLO

---

HARRY POTTER AND THE CHAMBER OF SECRETS

## Personalizzazione delle condizioni di errore

All'interno di un singolo trigger, è possibile stabilire condizioni d'errore differenti. Per ciascuna delle condizioni d'errore definite, è possibile selezionare un messaggio che compaia proprio quando si verifica l'errore. I numeri e i messaggi di errori visualizzati dall'utente sono impostati mediante la procedura RAISE\_APPLICATION\_ERROR, che può essere chiamata dall'interno di qualsiasi trigger.

Nell'esempio seguente, è riportato un trigger a livello di istruzione BEFORE DELETE sulla tabella BIBLIOTECA. Quando un utente cerca di cancellare un record della tabella BIBLIOTECA, questo trigger viene eseguito e verifica due condizioni di sistema: che il giorno della settimana non sia né sabato né domenica e che il nome utente Oracle dell'account che effettua l'eliminazione inizi con le lettere 'LIB'. I componenti del trigger sono descritti dopo il listato.

```

'SAT' or
  TO_CHAR(SysDate,'DY') = 'SUN' THEN
    RAISE weekend_error;
  end if;
  if SUBSTR(User,1,3) <> 'LIB' THEN
    RAISE not_library_user;

```

```

end if;
EXCEPTION
  WHEN weekend_error THEN
    RAISE_APPLICATION_ERROR (-20001,
      'Deletions not allowed on weekends');
  WHEN not_library_user THEN
    RAISE_APPLICATION_ERROR (-20002,
      'Deletions only allowed by Library users');
end;

```

L'intestazione definisce che si tratta di un trigger a livello di istruzione BEFORE DELETE:

```

create or replace trigger BIBLIOTECA_BEF_DEL
before delete on BIBLIOTECA

```

In questo trigger non compaiono clausole `when`, quindi il corpo del trigger verrà eseguito per tutte le cancellazioni.

La parte successiva del trigger dichiara i nomi delle due eccezioni definite all'interno del trigger stesso:

```

declare
  weekend_error EXCEPTION;
  not_library_user EXCEPTION;

```

La prima parte del corpo del trigger contiene una clausola `if` che utilizza la funzione `TO_CHAR` sulla funzione `SysDate`. Se il giorno corrente è sabato o domenica, verrà sollevata la condizione di errore `WEEKEND_ERROR`. Questa condizione di errore, detta *eccezione*, dev'essere definita all'interno del corpo del trigger.

```

if TO_CHAR(SysDate,'DY') = 'SAT' or
  TO_CHAR(SysDate,'DY') = 'SUN' THEN
  RAISE weekend_error;
end if;

```

Una seconda clausola `if` verifica che le prime tre lettere della pseudocolonna `User` siano `'LIB'`. Se il nome utente non inizia con `'LIB'`, verrà sollevata l'eccezione `NOT_LIBRARY_USER`. In questo esempio, viene utilizzato l'operatore `<>`, equivalente a `!=` (che significa "non è uguale").

```

if SUBSTR(Utente,1,3) <> 'LIB' THEN
  RAISE not_library_user;
end if;

```

La parte finale del corpo del trigger indica come gestire le eccezioni. Questa parte inizia con la parola chiave `exception`, seguita da una clausola `when` per ciascuna delle eccezioni. Tutte le eccezioni di questo trigger chiamano la procedura `RAISE_APPLICATION_ERROR`, che accetta due parametri di input: il numero dell'errore (che deve essere compreso tra -20001 e -20999) e il messaggio d'errore da visualizzare. In questo esempio, sono definiti due messaggi d'errore diversi, uno per ciascuna delle eccezioni definite:

```

EXCEPTION
  WHEN weekend_error THEN
    RAISE_APPLICATION_ERROR (-20001,
      'Deletions not allowed on weekends');

```

---

```
WHEN not_library_user THEN
  RAISE_APPLICATION_ERROR (-20002,
    'Deletions only allowed by Library users');
```

L'uso della procedura RAISE\_APPLICATION\_ERROR consente grande flessibilità nella gestione delle condizioni di errore che si possono incontrare all'interno del trigger. Per ulteriori informazioni sulle procedure, si rimanda al Capitolo 29.

**NOTA** *Le eccezioni verranno sollevate ugualmente anche se le operazioni di cancellazione non trovano righe da cancellare.*

Quando un utente non "LIB" cerca di cancellare una riga dalla tabella BIBLIOTECA, questo è il risultato ottenuto:

```
delete from BIBLIOTECA
where Titolo = 'MY LEDGER';

delete from BIBLIOTECA
*
ERROR at line 1:
ORA-20002: Deletions only allowed by Library users
ORA-06512: at "PRATICA.BIBLIOTECA_BEF_DEL", line 17
ORA-04088: error durino execution o trigger 'PRATICA.BIBLIOTECA_BEF_DEL'
```

## Chiamata di procedure all'interno dei trigger

Invece di creare un grande blocco di codice all'interno del corpo di un trigger, si può salvare il codice come una stored procedure e chiamare la procedura dall'interno del trigger con il comando call. Per esempio, se si crea una procedura INSERT\_BIBLIOTECA\_AUDIT\_DUP che inserisce delle righe in DUP\_AUDIT\_BIBLIOTECA, la si potrà chiamare da un trigger collocato sulla tabella AUDIT\_BIBLIOTECA, come mostrato nel listato seguente:

```
create or replace trigger BIBLIOTECA_AFT_INS_ROW
after insert on AUDIT_BIBLIOTECA
for each row
begin
call INSERT_BIBLIOTECA_AUDIT_DUP(:new.Titolo, :new.Editore,
:new.NomeCategoria, :new.Vecchia_Classificazione, :new.Nuova_Classificazione,
:new.Data_Audit);
end;
```

## Assegnazione dei nomi ai trigger

Il nome di un trigger deve indicare chiaramente la tabella cui esso si applica, i comandi DML che attivano il trigger, il suo stato before/after e se il trigger è a livello di riga o di istruzione. Dato che il nome di un trigger non può superare i 30 caratteri di lunghezza, quando si assegnano i nomi occorre utilizzare una serie standard di abbreviazioni. In generale, il nome del trigger deve comprendere una parte quanto più possibile ampia del nome della tabella. Pertanto, quando si crea un trigger a livello di riga BEFORE UPDATE, su una tabella di nome BIBLIOTECA\_PRESTITO non si dovrebbe assegnare al trigger il nome BEFORE\_UPDATE\_ROW\_LEVEL\_BP. Una soluzione migliore sarebbe BIBLIO\_PRESTITO\_BEF\_UPD\_ROW.

## Creazione di trigger di eventi DDL

È possibile creare dei trigger che vengono eseguiti quando si verifica un evento DDL. Se si pianifica di utilizzare raramente questa soluzione per motivi di sicurezza, si dovrebbe considerare la possibilità di usare il comando audit. Per esempio, si può usare il trigger di un evento DDL per eseguire il codice del trigger per i comandi create, alter e drop eseguiti su un cluster, una funzione, un indice, un package, una procedura, un ruolo, una sequenza, un sinonimo, una tabella, una tablespace, un trigger, un tipo, un utente oppure una vista. Se si usa la clausola on schema, il trigger verrà eseguito per qualsiasi nuovo oggetto del dizionario dati creato nello schema. Nell'esempio seguente, viene eseguita una procedura di nome INSERT\_AUDIT\_RECORD ogni volta che nello schema si creano degli oggetti:

```
create or replace trigger CREATE_DB_OBJECT_AUDIT
after create on schema
begin
    call INSERT_AUDIT_RECORD (ora_dict_obj_name);
end;
```

Come mostrato in questo esempio, è possibile fare riferimento ad attributi di sistema come il nome dell'oggetto. Gli attributi disponibili sono elencati nella Tabella 28.1.

Per proteggere gli oggetti all'interno di uno schema, si potrebbe creare un trigger che viene eseguito ogni volta che si cerca di eseguire il comando drop table. Dovrà essere un trigger di tipo BEFORE DROP:

```
create or replace trigger PREVENT_DROP
before drop on Pratica.schema
begin
    if ora_dict_obj_owner = 'PRATICA'
        and ora_dict_obj_name like 'B00%'
        and ora_dict_obj_type = 'TABELLA'
    then
        RAISE_APPLICATION_ERROR (
            -20002, 'Operation non permitted.');
    end if;
end;
```

Si osservi che il trigger fa riferimento agli attributi dell'evento all'interno delle sue clausole if. Il tentativo di eliminare una tabella nello schema di Pratica, il cui nome inizia con BIB ha come risultato:

```
drop table DUP_AUDIT_BIBLIOTECA;
drop table DUP_AUDIT_BIBLIOTECA
*
ERROR at line 1:
ORA-00604: error occurred at recursive SQL level 1
ORA-20002: Operation not permitted.
ORA-06512: at line 6
```

Per personalizzare ulteriormente il messaggio di errore visualizzato dall'utente, si può ricorrere alla procedura RAISE\_APPLICATION\_ERROR.

**Tabella 28.1** Attributi degli eventi di sistema.

ATTRIBUTO	TIPO	DESCRIZIONE	ESEMPIO
ora_client_ip_address	VARCHAR	Restituisce l'indirizzo IP del client in un evento LOGON, quando il protocollo di base è TCP/IP.	if (ora_sysevent = 'LOGON') then addr := ora_client_ip_address; end if;
ora_database_name	VARCHAR2(50)	Nome del database.	DECLARE db_name VARCHAR2(50); BEGIN db_name := ora_database_name; END;
ora_des_encrypted_password	VARCHAR2	La password con cifratura DES dell'utente che viene creata o modificata.	IF (ora_dict_obj_type = 'USER') THEN INSERT INTO event_table (ora_des_encrypted_password); END IF;
ora_dict_obj_name	VARCHAR(30)	Nome dell'oggetto del dizionario sul quale l'operazione ha avuto luogo.	INSERT INTO event_table (‘Changed object is ’    ora_dict_obj_name’);
ora_dict_obj_name_list (name_list OUT ora_name_list_t)	BINARY_INTEGER	Restituisce l'elenco dei nomi di oggetto degli oggetti modificati nell'evento.	if (ora_sysevent = 'ASSOCIATE STATISTICS') then number_modified := ora_dict_obj_name_list (name_list); end if;
ora_dict_obj_owner	VARCHAR(30)	Proprietario dell'oggetto del dizionario sul quale l'operazione ha avuto luogo.	INSERT INTO event_table ('object owner is'    ora_dict_obj_owner');
ora_dict_obj_owner_list (owner_list OUT ora_name_list_t)	BINARY_INTEGER	Restituisce l'elenco dei proprietari degli oggetti modificati nell'evento.	if (ora_sysevent = 'ASSOCIATE STATISTICS') then number_of_modified_objects := ora_dict_obj_owner_list (owner_list); end if;
ora_dict_obj_type	VARCHAR(20)	Tipo dell'oggetto del dizionario sul quale l'operazione ha avuto luogo.	INSERT INTO event_table ('This object is a '    ora_dict_obj_type');
ora_grantee( user_list OUT ora_name_list_t)	BINARY_INTEGER	Restituisce i beneficiari di un evento grant nel parametro OUT; restituisce il numero dei beneficiari nel valore restituito.	if (ora_sysevent = 'GRANT') then number_of_users := ora_grantee(user_list) end if;
ora_instance_num	NUMBER	Numero di istanza.	IF (ora_instance_num = 1) THEN INSERT INTO event_table ('1'); END IF;
ora_is.Alter.Column (Column_name IN VARCHAR2)	BOOLEAN	Restituisce TRUE se la colonna specificata è modificata.	if (ora_sysevent = 'ALTER' and ora_dict_obj_type = 'TABLE') then alter_column := ora_is.Alter.Column ('FOO'); end if;
ora_is_creating_nested_table	BOOLEAN	Restituisce TRUE se l'evento corrente sta creando una tabella annidata.	if (ora_sysevent = 'CREATE' and ora_dict_obj_type = 'TABLE' and ora_is_creating_nested_table) then insert into event_tab values (‘A nested table is created’); end if;

(segue)

**Tabella 28.1** Attributi degli eventi di sistema. (*continua*)

ATTRIBUTO	TIPO	DESCRIZIONE	ESEMPIO
ora_is_drop_column (Column_name IN VARCHAR2)	BOOLEAN	Restituisce TRUE se la colonna specificata viene eliminata.	if (ora_sysevent = 'ALTER' and ora_dict_obj_type = 'TABLE') then drop_column := ora_is_drop_column ('FOO'); end if;
ora_is_servererror	BOOLEAN	Restituisce TRUE se l'errore dato si trova sullo stack degli errori, altrimenti restituisce FALSE.	IF (ora_is_servererror (error_number)) THEN INSERT INTO event_table (‘Server error!!’); END IF;
ora_login_user	VARCHAR2(30)	Nome utente al login.	SELECT ora_login_user FROM dual;
ora_partition_pos	BINARY_INTEGER	In un trigger INSTEAD OF per CREATE TABLE, la posizione nel testo SQL nella quale si potrebbe inserire una clausola PARTITION.	-- Retrieve ora_sql_txt into -- sql_text variable first.  n := ora_partition_pos; new_stmt := substr(sql_text, 1, n-1)    ‘ ‘    my_partition_clause    ‘ ‘    substr(sql_text, n));
ora_privileges( Privilege_list OUT ora_name_list_t)	BINARY_INTEGER	Restituisce l'elenco di privilegi assegnati dal beneficiario o l'elenco dei privilegi revocati dal revocante. nel parametro OUT; restituisce il numero di privilegi nel valore restituito.	if (ora_sysevent = 'GRANT' or ora_sysevent = 'REVOKE') then number_of_privileges := ora_privileges(priv_list); end if;
ora_revoker (User_list OUT ora_name_list_t)	BINARY_INTEGER	Restituisce i revocanti di un evento di revoca nel parametro OUT; restituisce il numero dei revocanti nel valore restituito.	if (ora_sysevent = 'REVOKE') then number_of_users := ora_revoker(user_list);
ora_server_error	NUMBER	Data una posizione (1 per la cima dello stack), restituisce il numero dell'errore corrispondente alla posizione sullo stack degli errori.	INSERT INTO event_table ('top stack error'    ora_server_error(1));
ora_server_error_depth	BINARY_INTEGER	Restituisce il numero totale di messaggi di errore sullo stack degli errori.	n := ora_server_error_depth; -- This value is used with -- other functions such as -- ora_server_error
ora_server_error_msg (position in BINARY_INTEGER)	VARCHAR2	Data una posizione (1 per la cima dello stack), restituisce il messaggio dell'errore corrispondente alla posizione sullo stack degli errori.	INSERT INTO event_table ('top stack error message'    ora_server_error_msg(1));
ora_server_error_num_params (position in binary_integer)	BINARY_INTEGER	Data una posizione (1 per la cima dello stack), restituisce il numero delle stringhe che sono state sostituite nel messaggio di errore utilizzando un formato come "%s".	n := ora_server_error_num_params(1);

(segue)

**Tabella 28.1** Attributi degli eventi di sistema. (*continua*)

ATTRIBUTO	TIPO	DESCRIZIONE	ESEMPIO
ora_server_error_param (position in binary_integer, param in binary_integer)	VARCHAR2	Data una posizione (1 per la cima dello stack), e un numero di parametro, restituisce il valore di sostituzione "%s", "%d" e così via corrispondente nel messaggio di errore.	-- E.g. the 2nd %s in a message -- like "Expected %s, found %s" param := ora_server_error_param(1,2);
ora_sql_txt (sql_text out ora_name_list_t)	BINARY_INTEGER	Restituisce il testo SQL dell'istruzione che avvia il trigger nel parametro OUT. Se l'istruzione è lunga, viene suddivisa in più elementi di tabella PL/SQL. Il valore restituito dalla funzione specifica il numero di elementi nella tabella PL/SQL.	sql_text ora_name_list_t; stmt VARCHAR2(2000); ... n := ora_sql_txt(sql_text); FOR i IN 1..n LOOP stmt := stmt    sql_text(i); END LOOP; INSERT INTO event_table ('testo dell'istruzione che avvia il trigger: '    stmt);
ora_sysevent	VARCHAR2(20)	Evento di sistema che avvia il trigger: il nome dell'evento è uguale a quello nella sintassi.	INSERT INTO event_table (ora_sysevent);
ora_with_grant_option	BOOLEAN	Restituisce TRUE se i privilegi vengono concessi con l'opzione grant.	if (ora_sysevent = 'GRANT' and ora_with_grant_option = TRUE) then insert into event_table ('with grant option'); end if;
space_error_info( error_number OUT NUMBER, error_type OUT VARCHAR2, object_owner OUT VARCHAR2, table_space_ name OUT VARCHAR2, object_name OUT VARCHAR2, sub_object_ name OUT VARCHAR2)	BOOLEAN	Restituisce TRUE se l'errore è correlato a una condizione di spazio esaurito e compila il parametro OUT con informazioni relative all'oggetto che ha causato l'errore.	if (space_error_info(eno, typ, owner ts, obj, subobj) = TRUE) then  dbms_output.put_line('The object '    obj    ' owned by '    owner    ' has run out of space.');// end if;

## Creazione di trigger di eventi del database

Come gli eventi DML, anche gli eventi di database sono in grado di eseguire i trigger. Quando si verifica un evento del database (una chiusura, un avvio o un errore), è possibile eseguire un trigger che fa riferimento agli attributi dell'evento (come accade per gli eventi DML). Si potrebbe ricorrere al trigger di un evento di database per eseguire le funzioni di gestione del sistema subito dopo ogni avvio del database.

Per esempio, il trigger seguente fissa in memoria i package a ogni avvio del database. Il fissaggio dei package in memoria è un metodo efficiente per mantenere oggetti PL/SQL di gran-

di dimensioni in un shared pool di memoria, migliorando nel contempo le prestazioni e la stabilità del database. Il trigger PIN\_ON\_STARTUP, verrà eseguito ogni volta che si avvia il database.

```
create or replace trigger PIN_ON_STARTUP
after startup on database
begin
  DBMS_SHARED_POOL.KEEP (
    'SYS.STANDARD', 'P');
end;
```

Questo esempio mostra un trigger semplice che verrà eseguito subito dopo un avvio del database. L'elenco dei package nel corpo del trigger può essere modificato per includere quelli più utilizzati dalle applicazioni.

I trigger di avvio e chiusura possono accedere agli attributi ora\_instance\_num, ora\_database\_name, ora\_login\_user e ora\_sysevent elencati nella Tabella 28.1. Per la sintassi completa del comando create trigger, si rimanda al Capitolo 42.

## 28.5 Attivazione e disattivazione dei trigger

A differenza dei vincoli di integrità dichiarativa (come NOT NULL e PRIMARY KEY), i trigger non influiscono necessariamente su tutte le righe di una tabella. Essi agiscono solo sulle operazioni del tipo specificato, e solo mentre sono attivi. Qualsiasi dato generato prima della creazione di un trigger non verrà influenzato dal trigger stesso.

Per default, un trigger è attivato nel momento in cui viene creato, tuttavia ci sono situazioni in cui può essere necessario disattivare un trigger. Le due ragioni più comuni riguardano il caricamento dei dati. Durante caricamenti di dati voluminosi, potrebbe essere necessario disattivare i trigger che verrebbero eseguiti durante questa operazione. La disattivazione dei trigger durante i caricamenti dei dati può migliorare drasticamente le prestazioni del caricamento stesso. Una volta caricati i dati, si dovrà eseguire manualmente la manipolazione dei dati che sarebbe stata effettuata dal trigger se questo fosse stato attivo durante il caricamento dei dati.

La seconda ragione per la disattivazione di un trigger, sempre correlata al caricamento dei dati, interviene quando un caricamento di dati non viene completato con successo e dev'essere effettuato una seconda volta. In tal caso, è probabile che il caricamento dei dati sia riuscito parzialmente, quindi il trigger è stato eseguito per una parte dei record interessati. Durante un successivo caricamento, verrebbero inseriti gli stessi record. Pertanto, è possibile che lo stesso trigger venga eseguito due volte per la stessa operazione di inserimento (quando questa operazione si verifica durante i due caricamenti). In base alla natura delle operazioni e dei trigger, ciò potrebbe risultare errato. Se il trigger è stato attivato durante il caricamento non riuscito, sarà necessario disattivarlo prima di avviare un secondo processo di caricamento dei dati. Dopo che i dati sono stati caricati, si dovrà eseguire manualmente la manipolazione dei dati che sarebbe stata eseguita dal trigger se questo fosse stato attivo durante il caricamento dei dati.

Per attivare un trigger, occorre utilizzare il comando alter trigger con la parola chiave enable. Per usare questo comando, è necessario possedere la tabella oppure disporre del privilegio di sistema ALTER ANY TRIGGER. Nel listato seguente è riportato un comando alter trigger di esempio:

```
alter trigger BIBLIOTECA_BEF_UPD_INS_ROW enable;
```

Un secondo metodo per l'attivazione dei trigger utilizza il comando `alter table` con la clausola `enable all triggers`. Con questo comando non è possibile attivare trigger specifici; a tale fine si deve ricorrere al comando `alter trigger`. Nell'esempio seguente è descritto l'uso del comando `alter table`:

```
alter table BIBLIOTECA enable all triggers;
```

Per usare il comando `alter table`, è necessario possedere la tabella o disporre del privilegio di sistema `ALTER ANY TABLE`.

I trigger possono essere disattivati con gli stessi comandi di base (che richiedono gli stessi privilegi) apportando però qualche modifica alle rispettive clausole. Per il comando `alter trigger`, si utilizza la clausola `disable`.

```
alter trigger BIBLIOTECA_BEF_UPD_INS_ROW disable;
```

Per il comando `alter table`, si utilizza la clausola `disable all triggers`:

```
alter table BIBLIOTECA disable all triggers;
```

I trigger possono essere compilati a mano. Per compilare manualmente trigger già esistenti che non sono più validi, occorre utilizzare il comando `alter trigger compile`. Dal momento che i trigger dipendono da altri elementi, possono divenire non validi se un oggetto su cui si basa il trigger viene modificato. Il comando `alter trigger debug` consente la generazione di informazioni PL/SQL durante la ricompilazione del trigger.

## 28.6 Sostituzione di trigger

L'unica parte che può essere modificata è lo stato di un trigger. Per modificare il corpo di un trigger, quest'ultimo dovrà essere ricreato o sostituito. Quando si sostituisce un trigger, si utilizza il comando `create or replace trigger` (fare riferimento al paragrafo “Sintassi dei trigger” e agli esempi riportati in precedenza nel capitolo).

## 28.7 Eliminazione di trigger

I trigger possono essere eliminati mediante il comando `drop trigger`. Per eliminare un trigger, è necessario possedere il trigger o disporre del privilegio di sistema `DROP ANY TRIGGER`. Di seguito è riportato un esempio di questo comando:

```
drop trigger BIBLIOTECA_BEF_UPD_INS_ROW;
```



## Capitolo 29

# Procedure, funzioni e package

- 29.1 **Privilegi di sistema richiesti**
- 29.2 **Privilegi di tabella richiesti**
- 29.3 **Procedure e funzioni**
- 29.4 **Procedure e package**
- 29.5 **Sintassi del comando create procedure**
- 29.6 **Sintassi del comando create function**
- 29.7 **Sintassi del comando create package**
- 29.8 **Visualizzazione del codice sorgente di oggetti procedurali**
- 29.9 **Compilazione di procedure, funzioni e package**
- 29.10 **Sostituzione di procedure, funzioni e package**
- 29.11 **Eliminazione di procedure, funzioni e package**

In Oracle, è possibile memorizzare come *procedure* sia sofisticate regole business sia la logica applicativa. Le *stored procedure*, gruppi di istruzioni SQL, PL/SQL e Java, consentono di spostare il codice che impone regole legate all'attività dall'applicazione al database. In questo modo, il codice verrà memorizzato una sola volta per essere utilizzato più volte. Dato che Oracle supporta le stored procedure, il codice all'interno delle applicazioni diviene più coerente e facile da gestire.

**NOTA** Per dettagli su Java e il suo impiego nelle stored procedure, si rimanda ai Capitoli 34, 35 e 36. Questo capitolo si concentrerà sulle procedure PL/SQL.

Le procedure e gli altri comandi PL/SQL possono essere raggruppati in *package*. Nei paragrafi successivi verranno forniti i dettagli di implementazione e alcune raccomandazioni riguardanti i package, le procedure e le *funzioni* (procedure che possono restituire valori all'utente).

L'uso di procedure può assicurare miglioramenti delle prestazioni per due motivi.

- L'elaborazione di regole business complesse può essere effettuata all'interno del database, quindi mediante il server. Nelle applicazioni client-server, o a tre livelli, lo spostamento di elaborazioni complesse dall'applicazione (sul client) al database (sul server) può migliorare sensibilmente le prestazioni.
- Dato che il codice procedurale è memorizzato all'interno del database ed è piuttosto statico, è anche possibile trarre vantaggio dal riutilizzo delle stesse query all'interno del database. L'area

SQL condivisa (Shared SQL Area) della SGA (System Global Area) memorizza le versioni analizzate dei comandi eseguiti. Così una procedura, la seconda volta che viene eseguita, può sfruttare l'analisi effettuata in precedenza, migliorando le prestazioni nell'esecuzione.

Oltre a ciò, anche lo sviluppo di applicazioni può trarre dei vantaggi. Le regole business consolidate all'interno del database non dovranno più essere scritte in ciascuna applicazione, il che consente risparmi di tempo durante la creazione delle applicazioni stesse e semplifica il processo di gestione.

## 29.1 Privilegi di sistema richiesti

Per creare un oggetto procedurale, è necessario disporre del privilegio di sistema CREATE PROCEDURE (che fa parte del ruolo RESOURCE). Se l'oggetto procedurale si trova nello schema di un altro utente, è necessario disporre del privilegio di sistema CREATE ANY PROCEDURE.

### Esecuzione delle procedure

Un oggetto procedurale, una volta creato, può essere eseguito. Quando si esegue un oggetto procedurale, questo può fare riferimento ai privilegi di tabella del suo proprietario, oppure ai privilegi dell'utente che lo esegue. Quando la procedura viene creata con i privilegi del proprietario, non è necessario concedere a un utente che esegue questa procedura l'accesso alle tabelle cui accede la procedura in questione.

Nelle versioni precedenti di Oracle, gli oggetti procedurali potevano essere eseguiti solo con i privilegi del proprietario della procedura (definiti *privilegi del proprietario*). In alternativa, si possono usare i *privilegi dell'utente chiamante*, nel qual caso le procedure vengono eseguite con i privilegi dell'utente che esegue la procedura. Se una procedura utilizza i privilegi dell'utente invocante, l'utente dovrà avere accesso a tutti gli oggetti del database cui accede anche la procedura. A meno che non sia specificato diversamente, gli esempi di questo capitolo presuppongono che gli utenti esegano le procedure con i privilegi del proprietario.

Per consentire agli altri utenti di eseguire un oggetto procedurale, occorre concedere loro il privilegio EXECUTE per quell'oggetto, come mostrato nell'esempio seguente:

```
grant execute on MIA_PROCEDURA to Dora;
```

Ora l'utente Dora sarà in grado di eseguire la procedura denominata MIA\_PROCEDURA. Se non si concede il privilegio EXECUTE agli altri utenti, per eseguire la procedura questi dovranno disporre del privilegio di sistema EXECUTE ANY PROCEDURE.

Durante l'esecuzione delle procedure, normalmente a queste ultime vengono passate delle variabili. Per esempio, una procedura che interagisce con la tabella BIBLIOTECA può accettare un valore della colonna Titolo come input. In questo esempio, si parte dal presupposto che la procedura crea un nuovo record nella tabella BIBLIOTECA quando si aggiunge un nuovo libro (dall'elenco dei libri contenuto nella tabella LIBRO\_ORDINE). Questa procedura può essere chiamata da qualsiasi applicazione all'interno del database (posto che l'utente chiamante dispone del privilegio EXECUTE per la procedura in questione).

La sintassi utilizzata per eseguire una procedura dipende dall'ambiente da cui la procedura viene chiamata. Dall'interno di SQL\*Plus, una procedura può essere eseguita con il comando execute, seguito dal nome della procedura stessa. Qualsiasi argomento da passare alla procedura

dev'essere racchiuso tra parentesi e seguire il nome della procedura, come nell'esempio seguente (che utilizza la procedura NUOVO\_LIBRO):

```
execute NUOVO_LIBRO('ONCE REMOVED');
```

Il comando eseguirà la procedura NUOVO\_LIBRO, passandole il valore ONCE REMOVED.

Dall'interno di un'altra procedura, funzione, package o trigger, la procedura può essere chiamata senza il comando execute. Se la procedura NUOVO\_LIBRO è stata chiamata da un trigger sulla tabella LIBRO\_ORDINE, il corpo di questo trigger può includere il comando seguente:

```
NUOVO_LIBRO(:nuovo.Titolo);
```

La procedura NUOVO\_LIBRO verrà eseguita con il nuovo valore della colonna Titolo come input (per ulteriori informazioni sull'uso di valori vecchi e nuovi all'interno dei trigger, si rimanda al Capitolo 28).

Per eseguire una procedura posseduta da un altro utente, è necessario creare un sinonimo per questa procedura, oppure fare riferimento al nome del proprietario durante l'esecuzione, come mostrato nel listato seguente:

```
execute Pratica.NUOVO_LIBRO('ONCE REMOVED');
```

Il comando eseguirà la procedura NUOVO\_LIBRO nello schema di Pratica.

In alternativa, si potrebbe creare un sinonimo per la procedura con il comando seguente:

```
create synonym NUOVO_LIBRO for Pratica.NUOVO_LIBRO;
```

Quindi il proprietario di questo sinonimo non dovrebbe più fare riferimento al proprietario della procedura per esegirla. Sarebbe sufficiente inserire il comando:

```
execute NUOVO_LIBRO('ONCE REMOVED');
```

e il sinonimo punterebbe alla procedura corretta.

Per l'esecuzione di procedure remote, è necessario specificare il nome di un database link (per informazioni sui database link, si rimanda al Capitolo 22). Il nome del database link dev'essere specificato subito dopo il nome della procedura e precedere quello delle variabili, come illustrato nell'esempio seguente:

```
execute NUOVO_LIBRO@REMOTE_CONNECT('ONCE REMOVED');
```

Il comando si serve del database link REMOTE\_CONNECT per accedere alla procedura NUOVO\_LIBRO in un database remoto.

Per semplificare l'individuazione della procedura da parte dell'utente, è possibile creare un sinonimo per la procedura remota, come nell'esempio seguente:

```
create synonym NUOVO_LIBRO
  for NUOVO_LIBRO@REMOTE_CONNECT;
```

Una volta creato il sinonimo, l'utente potrà fare riferimento alla procedura remota grazie al nome del sinonimo. Oracle assume che tutte le chiamate alla procedura remota richiedano aggiornamenti nel database remoto.

## 29.2 Privilegi di tabella richiesti

Gli oggetti procedurali possono fare riferimento a tabelle. Per fare in modo che gli oggetti vengano eseguiti correttamente, il proprietario della procedura, del package o della funzione da eseguire deve disporre di privilegi sulle tabelle utilizzate. A meno che non si utilizzino i privilegi dell’utente chiamante, l’utente che esegue l’oggetto procedurale non ha bisogno di privilegi sulle tabelle di base.

**NOTA** *I privilegi necessari per procedure, package e funzioni non possono derivare da ruoli, ma devono essere concessi direttamente al proprietario della procedura, del package o della funzione.*

## 29.3 Procedure e funzioni

Le funzioni sono in grado di restituire un valore all’utente chiamante, e nelle query è possibile fare un riferimento diretto alle funzioni. Questo valore viene restituito utilizzando la parola chiave return all’interno della funzione. Esempi di funzioni sono riportati nel paragrafo “Sintassi del comando create function”, più avanti in questo capitolo.

## 29.4 Procedure e package

I package sono gruppi di procedure, funzioni, variabili e istruzioni SQL riuniti in un’unica unità. Per eseguire una procedura all’interno di un package, per prima cosa occorre elencare il nome del package, seguito dal nome della procedura, come mostrato nell’esempio seguente:

```
execute INVENTARIO_LIBRI.NUOVO_LIBRO('ONCE REMOVED');
```

In questo caso, è stata eseguita la procedura NUOVO\_LIBRO all’interno del package INVENTARIO\_LIBRI.

I package consentono a più procedure di utilizzare le stesse variabili e gli stessi cursori. Le procedure all’interno di package possono essere disponibili al pubblico (come la procedura NUOVO\_LIBRO nell’esempio precedente) o private, nel qual caso sarà possibile accedervi solo mediante comandi dall’interno del package (come chiamate da altre procedure). Esempi di package sono mostrati nel paragrafo “Sintassi del comando create package”, più avanti nel capitolo.

I package possono anche includere comandi da eseguire ogniqualvolta il package viene chiamato, a prescindere dalla procedura o funzione chiamata all’interno del package. Pertanto, i package non solo raggruppano procedure, ma consentono anche di eseguire comandi che non siano specifici delle procedure raggruppate. Per un esempio di codice eseguito ogni volta che un package viene chiamato, si rimanda al paragrafo “Inizializzazione dei package” più avanti nel capitolo.

## 29.5 Sintassi del comando create procedure

La sintassi del comando create procedure è riportata nel Capitolo 42. Questa sintassi è:

---

```

create [or replace] procedure [schema .] procedura
[( argomento [ in | out | in out ] [nocopy] tipodati
  [. argomento [ in | out | in out ] [nocopy] tipodati]...
  )]
[authid { current_user | definier }]
{ is | as } { corpo_sottoprogramma_sq1 |
  language { java name 'stringa'| c [name nome] library nome_lib
  [agent in ( argomento )]
  [with context] [parameters ( parametri )]
}};


```

Con questo comando vengono creati sia l'intestazione sia il corpo della procedura.

La procedura NUOVO\_LIBRO viene creata dal comando mostrato nel listato seguente:

```

create or replace procedure NUOVO_LIBRO (aTitolo IN VARCHAR2,
  aEditore IN VARCHAR2, aNomeCategoria IN VARCHAR2)
as
begin
  insert into BIBLIOTECA (Titolo, Editore, NomeCategoria, Classificazione)
    values (aTitolo, aEditore, aNomeCategoria, NULL);
  delete from LIBRO_ORDINE
    where Titolo = aTitolo;
end;


```

In questo caso, la procedura NUOVO\_LIBRO accetta come input il titolo, l'editore e la categoria di un libro. Questa procedura potrà essere chiamata da qualsiasi applicazione. Inserisce un record nella tabella BIBLIOTECA con un valore NULL per la colonna Classificazione e cancella il Titolo corrispondente dalla tabella LIBRO\_ORDINE.

È possibile sostituire una procedura già esistente mediante il comando `create or replace procedure`. Il vantaggio derivato dall'uso di questo comando (invece di eliminare e ricreare la vecchia procedura) è la conservazione delle concessioni di privilegi EXECUTE effettuate in precedenza per la procedura.

Il qualificatore IN viene utilizzato per gli argomenti per i quali è necessario specificare i valori quando si chiama la procedura. Nell'esempio di NUOVO\_LIBRO, gli argomenti aTitolo, aEditore e aNomeCategoria sono dichiarati come IN. Il qualificatore OUT segnala che la procedura restituisce un valore all'utente chiamante tramite questo argomento. Il qualificatore IN OUT significa che l'argomento è sia IN sia OUT: quando si chiama la procedura è necessario specificare un valore per questo argomento; la procedura restituirà poi un valore al chiamante attraverso l'argomento stesso. Se non viene specificato un tipo di qualificatore, l'impostazione predefinita è IN.

Nella sintassi di `create procedure`, authid si riferisce al tipo di autenticazione: privilegi del proprietario (impostazione predefinita) o privilegi dell'utente chiamante (`current_user`). La parola chiave `nocopy` indica a Oracle di restituire all'utente il valore della variabile il più velocemente possibile.

Per default, una procedura è costituita da un blocco di codice scritto con PL/SQL (*blocco* si riferisce al codice che la procedura esegue una volta chiamata). Nell'esempio NUOVO\_LIBRO, il blocco è:

```

begin
  insert into BIBLIOTECA (Titolo, Editore, NomeCategoria, Classificazione)
    values (aTitolo, aEditore, aNomeCategoria, NULL);
  delete from LIBRO_ORDINE
    where Titolo = aTitolo;
end;


```

Questo blocco PL/SQL è abbastanza semplice, essendo costituito da un'unica istruzione SQL. I blocchi PL/SQL all'interno di procedure possono comprendere qualsiasi istruzione DML, ma non possono essere utilizzati per istruzioni DDL (come `create view`).

In alternativa, `language` fa riferimento al linguaggio con cui è stato scritto il codice. Per esempi relativi a Java, si rimanda al Capitolo 36.

Il listato seguente illustra l'esecuzione della procedura NUOVO\_LIBRO, e le conseguenti modifiche apportate alle tabelle LIBRO\_ORDINE e BIBLIOTECA.

```
select Titolo from LIBRO_ORDINE;

TITOLO
-----
SHOELESS JOE
GOSPEL
SOMETHING SO STRONG
GALILEO'S DAUGHTER
LONGITUDE
ONCE REMOVED

execute NUOVO_LIBRO('ONCE REMOVED','SANCTUARY PUB','SAGGIADULTI');

PL/SQL procedure successfully completed.
```

```
select Titolo from LIBRO_ORDINE;

TITOLO
-----
SHOELESS JOE
GOSPEL
SOMETHING SO STRONG
GALILEO'S DAUGHTER
LONGITUDE

select * from BIBLIOTECA
where Titolo = 'ONCE REMOVED';

TITOLO
-----  
EDITORE      NOME CATEGORIA      RA
-----  
ONCE REMOVED          SANCTUARY PUB      SAGGIADULTI
```

## 29.6 Sintassi del comando `create function`

La sintassi del comando `create function`, è più complessa della sintassi per il comando `create procedure`. A un livello alto la sintassi è:

```
create [or replace] function [schema .] funzione
[ ( argomento [ in | out | in out ] [nocopy] tipodati
  [, argomento [ in | out | in out ] [nocopy] tipodati]...
) ]
```

```

return tipodati
[ { clausola_privilegi_utentechiamante | deterministic | clausola_abilitazione_parallelala }
  [clausola_privilegi_utentechiamante | deterministic | clausola_abilitazione_parallelala]...
]
{ { aggregate | pipelined } using [schema .] tipo_implementazione
| [pipelined] { is | as } { corpo_funzione_pl/sql | spec_chiamata }};
```

Con questo comando vengono creati sia l'intestazione sia il corpo della funzione.

La parola chiave `return` specifica il tipo di dati del valore restituito dalla funzione. Questo può essere qualunque tipo di dati PL/SQL valido (si consulti il capitolo 27). Ogni funzione deve comprendere una clausola `return`, in quanto una funzione deve per definizione restituire un valore all'ambiente che la chiama.

L'esempio seguente mostra una funzione di nome `SPESE_RITARDO`, che restituisce per ogni persona le spese dovute a ritardi nella riconsegna dei libri, basandosi su calcoli eseguiti sulla tabella `BIBLIOTECA_PRESTITO`. L'input è il nome della persona mentre l'output è il saldo relativo alla stessa persona.

```

create or replace function SPESE_RITARDO (aNome IN VARCHAR2)
  return NUMBER
  is
    saldo NUMBER(10,2);
begin
  select SUM((DataRestituzione-DataPrestito) -14)*0.20
  into saldo
  from BIBLIOTECA_PRESTITO
  where Nome = aNome;
  RETURN(saldo);
end;
```

Per mostrare le differenze tra procedure e funzioni, si esamini questa funzione pezzo per pezzo. Per prima cosa, viene assegnato un nome alla funzione, quindi viene specificato l'input:

```
create or replace function SPESE_RITARDO (aNome IN VARCHAR2)
```

In seguito vengono definite le caratteristiche del valore da restituire. La definizione della variabile il cui valore verrà restituito è costituita da tre parti: il tipo di dati, il nome e la lunghezza. In questo esempio, il tipo di dati viene impostato tramite la clausola `return NUMBER`. Alla variabile viene quindi assegnato il nome “`saldo`”, e viene definita come `NUMBER(10,2)`. In questo modo sono state definite tutte e tre le parti della variabile: il suo tipo di dati, il suo nome e la sua lunghezza.

```

  return NUMBER
  is
    saldo NUMBER(10,2);
```

Segue il blocco PL/SQL. Nell'istruzione SQL, viene calcolata la somma di tutte le tasse di ritardo (€0.20 al giorno una volta superati 14 giorni). La somma viene selezionata in una variabile di nome `saldo` (definita precedentemente). La linea di comando `RETURN(saldo)` restituisce quindi il valore della variabile “`saldo`” al programma chiamante.

```

begin
  select SUM((DataRestituzione_DataPrestito) -14)*0.20
  into saldo
  from BIBLIOTECA_PRESTITO
```

```

    where Nome = aNome;
    RETURN(saldo);
end;

```

È possibile sostituire una funzione già esistente mediante il comando `create or replace function`. Con la clausola `or replace`, qualsiasi privilegio `EXECUTE` preventivamente concesso alla funzione verrà mantenuto.

Per creare la funzione in un account diverso (detto anche *schema*), è necessario disporre del privilegio di sistema `CREATE ANY PROCEDURE`. Se non è specificato uno schema, la funzione verrà creata nello schema di colui che la crea. Per creare una funzione nel proprio schema, è necessario disporre del privilegio di sistema `CREATE PROCEDURE` (che fa parte del ruolo `RESOURCE`). Il privilegio di creare procedure comprende anche il privilegio di creare funzioni e package.

È possibile controllare la funzione creando una variabile e impostandone il valore uguale al valore restituito dalla funzione:

```

variable saldo number;
execute :saldo := SPESA_RITARDO('FRED FULLER');

```

PL/SQL procedure successfully completed.

Per verificare i risultati, è sufficiente visualizzare il valore della variabile:

```

print saldo

```

SALDO
-----
3.8

## Riferimenti a tavole remote nelle procedure

L'accesso a tavole remote è consentito dalle istruzioni SQL nelle procedure. È possibile eseguire una query su una tabella remota tramite un database link della procedura, come mostrato nell'esempio seguente. In questo esempio, la procedura `NUOVO_LIBRO` inserisce un record nella tabella `BIBLIOTECA` nel database definito dal database link `REMOTE_CONNECT` mentre esegue cancellazioni da una copia locale di `LIBRO_ORDINE`.

```

create or replace procedure NUOVO_LIBRO (aTitolo IN VARCHAR2,
                                         aEditore IN VARCHAR2, aNomeCategoria IN VARCHAR2)
as
begin
    insert into BIBLIOTECA@REMOTE_CONNECT
        (Titolo, Editore, NomeCategoria, Classificazione)
        values (aTitolo, aEditore, aNomeCategoria, NULL);
    delete from LIBRO_ORDINE
        where Titolo = aTitolo;
end;

```

Le procedure possono utilizzare anche dei sinonimi locali. Per esempio, si potrebbe creare un sinonimo locale per una tabella remota, come nell'esempio seguente:

```
create synonym BIBLIOTECA for BIBLIOTECA@REMOTE_CONNECT;
```

Sarà quindi possibile riscrivere la procedura per rimuovere le specifiche del database link:

```
create or replace procedure NUOVO_LIBRO (aTitolo IN VARCHAR2,
                                         aEditore IN VARCHAR2, aNomeCategoria IN VARCHAR2)
as
begin
    insert into BIBLIOTECA
        (Titolo, Editore, NomeCategoria, Classificazione)
        values (aTitolo, aEditore, aNomeCategoria, NULL);
    delete from LIBRO_ORDINE
        where Titolo = aTitolo;
end;
```

La rimozione dei nomi dei database link dalle procedure consente di rimuovere dalla procedura stessa anche i dettagli riguardanti la collocazione fisica della tabella. Se la collocazione della tabella cambia, verrà modificato solo il sinonimo, mentre la procedura resterà valida.

## Ricerca degli errori nelle procedure

Il comando SQLPLUS show errors visualizza tutti gli errori associati all'oggetto procedurale creato per ultimo. Questo comando cerca nella vista del dizionario dati USER\_ERRORS gli errori associati al tentativo di compilazione più recente per l'oggetto procedurale in esame. Il comando show errors visualizza il numero di linea e di colonna per ciascun errore, insieme al testo del messaggio di errore.

Per visualizzare gli errori associati a oggetti procedurali creati in precedenza, è possibile eseguire una query diretta su USER\_ERRORS, come mostrato nel listato seguente. Questo esempio esegue una query su USER\_ERRORS per trovare i messaggi d'errore incontrati durante la creazione della funzione SPESE\_RITARDO, descritta in precedenza in questo capitolo.

```
select Linea,      /*Numero di linea dell'errore. */
       Posizione, /*Numero di colonna dell'errore.*/
       Testo      /*Testo del messaggio di errore.*/
  from USER_ERRORS
 where Nome = 'SPESE_RITARDO'
   and Tipo = 'FUNCTION'
 order by Sequenza;
```

Valori validi per la colonna Tipo sono VIEW, PROCEDURE, PACKAGE, FUNCTION, e PACKAGE BODY.

Per recuperare informazioni su errori relativi a oggetti procedurali, si possono utilizzare anche altri due livelli di viste del dizionario dati, ALL e DBA. Per informazioni su tali viste, si rimanda al Capitolo 37.

## Uso del package DBMS\_OUTPUT

Oltre alle informazioni relative agli errori fornite dal comando show errors, si potrebbe usare il package DBMS\_OUTPUT, appartenente a un gruppo di package installati automaticamente quando si crea un database Oracle.

Per usare DBMS\_OUTPUT, occorre eseguire il comando set serveroutput on prima di eseguire l'oggetto procedurale sui cui si effettuerà il debug.

DBMS\_OUTPUT consente di utilizzare tre funzioni di debug all'interno del package:

---

PUT	Colloca più risultati sulla stessa linea.
PUT_LINE	Colloca ciascun risultato su una linea separata.
NEW_LINE	Utilizzata con PUT; segnala la fine della linea di output corrente.

---

PUT e PUT\_LINE vengono utilizzate per generare le informazioni di debug che si desidera visualizzare. Per esempio, se si effettua il debug di una procedura che include un ciclo (fare riferimento al Capitolo 27), è possibile tenere traccia delle modifiche in una variabile a ogni esecuzione successiva del ciclo. Per tenere traccia dei valori di una variabile, è possibile utilizzare un comando simile a quello descritto nel listato seguente. In questo esempio, viene visualizzato il valore della colonna Saldo preceduto dalla stringa letterale ‘Dovuto:’

```
DBMS_OUTPUT.PUT_LINE('Dovuto: ' || saldo);
```

PUT e PUT\_LINE possono essere utilizzate anche al di fuori dei cicli, ma con l’uso del comando RETURN nelle funzioni si possono ottenere gli stessi risultati in modo migliore (si consulti il paragrafo “Sintassi del comando create function” introdotto in precedenza nel capitolo).

## Creazione di funzioni personalizzate

Invece di limitarsi a chiamare le funzioni personalizzate tramite i comandi execute, le si potrebbe utilizzare all’interno di espressioni SQL. Ciò consente di ampliare la funzionalità di SQL, adattandolo alle proprie esigenze. Queste funzioni personalizzate possono essere utilizzate nello stesso modo in cui si utilizzano le funzioni messe a disposizione da Oracle, come SUBSTR e TO\_CHAR. Le funzioni personalizzate non possono essere utilizzate nei vincoli CHECK o DEFAULT e non sono in grado di manipolare valori di database.

Si possono chiamare funzioni indipendenti (create con il comando create function descritto nei paragrafi precedenti) oppure funzioni dichiarate nelle specifiche di package (si rimanda al paragrafo “Sintassi del comando create package” più avanti nel capitolo). Le procedure non vengono chiamate direttamente da SQL, ma possono essere chiamate da funzioni personalizzate.

Per esempio, si consideri la funzione SPESE\_RITARDO presentata più indietro nel capitolo, che calcolava le spese dovute a ritardi nella restituzione dei libri. Tale funzione aveva una sola variabile di input: il nome della persona. Tuttavia, per visualizzare i risultati di SPESE\_RITARDO per tutte le persone che hanno preso un libro in prestito, in genere si dovrebbe eseguire la procedura una volta per ciascun record contenuto nella tabella BIBLIOTECA\_PRESTITO.

Il processo per il calcolo delle spese di ritardo può essere migliorato. Si consideri la query riportata di seguito:

```
create view UTENTI_PRESTITO_LIBRI
as select distinct Nome from BIBLIOTECA_PRESTITO;

select Nome,
       SPESE_RITARDO(Nome)
  from UTENTI_PRESTITO_LIBRI;
```

Oracle restituisce questo output:

---

NOME	SPESE_RITARDO(NOME)
DORAH TALBOT	.8
EMILY TALBOT	.2
FRED FULLER	3.8
GERHARDT KENTGEN	2.8
JED HOPKINS	1.4
PAT LAVAY	2.6
ROLAND BRANDT	22.6

7 rows selected.

Questa query singola si serve della funzione personalizzata SPESE\_RITARDO per calcolare le spese di ritardo per tutte le persone che hanno preso in prestito un libro. Non è esattamente così: le persone ricevono dei crediti per i libri che hanno restituito in anticipo. Per aggiungere una clausola `where` alla query della funzione, è possibile utilizzare il comando `create or replace function`:

```
create or replace function SPESE_RITARDO (aNome IN VARCHAR2)
  return NUMBER
  is
    saldo NUMBER(10,2);
begin
  select SUM((DataRestituzione-DataPrestito) -14)*0.20
  into saldo
  from BIBLIOTECA_PRESTITO
  where Nome = aNome
        and DataRestituzione-DataPrestito > 14;
  RETURN(saldo);
end;
```

L'output corretto è visualizzato nel listato seguente:

```
select Nome,
       SPESE_RITARDO(Nome)
  from UTENTI_PRESTITO_LIBRI;
```

---

NOME	SPESE_RITARDO(NOME)
DORAH TALBOT	.4
EMILY TALBOT	.8
FRED FULLER	3.8
GERHARDT KENTGEN	4.6
JED HOPKINS	1.4
PAT LAVAY	3.4
ROLAND BRANDT	22.6

La query ha selezionato ciascun nome dalla tabella UTENTI\_PRESTITO\_LIBRI; per ciascun nome ha eseguito la funzione SPESE\_RITARDO che ha selezionati i dati da BIBLIOTECA\_PRESTITO.

Per sfruttare questa caratteristica, le funzioni personalizzate devono seguire le stesse linee guida delle funzioni di Oracle. In particolare, non devono aggiornare il database e devono contenere solo parametri `IN`.

**NOTA** Anche se questa tecnica funziona, comporta un calo delle prestazioni. Maggiore è il numero dei dati nella tabella, maggiore sarà il calo delle prestazioni quando si confronterà questo metodo con un join tradizionale.

## Personalizzazione di condizioni di errore

All'interno degli oggetti procedurali è possibile stabilire condizioni di errore differenti (per esempi di condizioni di errore personalizzate all'interno di trigger, fare riferimento al Capitolo 28). Per ciascuna delle condizioni di errore definite, è possibile scegliere un messaggio di errore che verrà visualizzato al verificarsi dell'errore stesso. I numeri e i messaggi degli errori visualizzati dall'utente sono impostati mediante la procedura RAISE\_APPLICATION\_ERROR, che può essere chiamata dall'interno di qualsiasi oggetto procedurale.

La procedura RAISE\_APPLICATION\_ERROR può essere chiamata dall'interno di procedure, package e funzioni. Essa richiede due valori di input: il numero e il testo del messaggio. Sia il numero sia il testo del messaggio che verrà visualizzato dall'utente possono essere impostati. Questa rappresenta un'aggiunta molto potente alle eccezioni standard disponibili nel linguaggio PL/SQL (si consulti il paragrafo “Eccezioni” del Capitolo 42).

Nell'esempio seguente è riportata la funzione SPESE\_RITARDO descritta in precedenza nel capitolo. Ora però presenta una sezione in più (scritta in grassetto), intitolata EXCEPTION, che indica a Oracle come gestire le elaborazioni non standard. In questo esempio, il messaggio standard di eccezione NO\_DATA\_FOUND viene ridefinito per mezzo della procedura RAISE\_APPLICATION\_ERROR.

```
create or replace function SPESE_RITARDO (aNome IN VARCHAR2)
  return NUMBER
  is
    saldo NUMBER(10,2);
begin
  select SUM((DataRestituzione-DataPrestito) -14)*0.20
  into saldo
  from BIBLIOTECA_PRESTITIO
  where Nome = aNome
    and (DataRestituzione-DataPrestito) > 14;
  RETURN(saldo);
EXCEPTION
when NO_DATA_FOUND THEN
  RAISE_APPLICATION_ERROR(-20100,
  'No books borrowed.');
end;
```

Nell'esempio precedente, viene utilizzata l'eccezione NO\_DATA\_FOUND. Se si desidera definire eccezioni personalizzate, è necessario nominare l'eccezione in una sezione di dichiarazioni della procedura, che precede immediatamente il comando begin. Come mostrato nel listato seguente, questa sezione dovrebbe includere voci per ciascuna delle eccezioni personalizzate definite, elencate come tipo EXCEPTION:

```
declare
some_custom_error EXCEPTION;
```

**NOTA** Se si usano le eccezioni già definite all'interno del linguaggio PL/SQL, non è necessario elencarle nella sezione delle dichiarazioni dell'oggetto procedurale. Per un elenco delle eccezioni predefinite si rimanda al paragrafo “Eccezioni” del Capitolo 42.

Nella parte exception del codice relativo all'oggetto procedurale, si specifica al database come gestire le eccezioni. Questa parte inizia con la parola chiave exception, seguita da una clausola WHEN per ciascuna eccezione. Ogni eccezione può chiamare la procedura RAISE\_APPLICATION\_ERROR, che accetta due parametri di input: il numero dell'errore (che dev'essere compreso tra -20001 e -20999) e il messaggio d'errore da visualizzare. Nell'esempio precedente era definita una sola eccezione. Tuttavia, si possono definire più eccezioni, come mostrato nel listato seguente; si può ricorrere alla clausola when others per gestire tutte le eccezioni non specificate:

```
EXCEPTION
when NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20100,
    'No books borrowed.');
when some_custom_error THEN
    RAISE_APPLICATION_ERROR (-20101,
    'Some custom error message.');
```

L'uso della procedura RAISE\_APPLICATION\_ERROR conferisce grande flessibilità nella gestione delle condizioni di errore nelle quali ci si può imbattere all'interno di oggetti procedurali.

## Denominazione di procedure e funzioni

Alle procedure e alle funzioni dovrebbe essere assegnato un nome coerente con la funzione da esse svolta o con le regole da esse imposte. Non dovrebbe esserci ambiguità sui loro scopi.

Alla procedura NUOVO\_LIBRO descritta precedentemente è necessario assegnare un nuovo nome. La procedura NUOVO\_LIBRO svolge una funzione gestionale: inserisce i record nella tabella BIBLIOTECA e li cancella da LIBRO\_ORDINE, quindi il suo nome dovrebbe rispecchiare tale funzione. Un nome migliore potrebbe essere RICEVE\_ORDINE\_LIBRO. Dato che la procedura esegue una funzione, un verbo (in questo caso "riceve") deve descrivere ciò che la procedura fa. Il nome della procedura dovrebbe contenere anche il nome delle principali tabelle su cui agisce. Se le tabelle sono nominate in modo corretto, allora il nome della tabella dovrebbe essere l'oggetto su cui agisce direttamente il verbo (in questo caso, LIBRO\_ORDINE o BIBLIOTECA). Per motivi di coerenza, si continuerà a riferirsi alla procedura con il nome NUOVO\_LIBRO per la parte restante del capitolo.

## 29.7 Sintassi del comando create package

Quando si creano i package, la specifica e il corpo del package vengono creati separatamente. Per questo motivo, i comandi da utilizzare sono due: create package per la specifica del package, e create package body per il corpo. Entrambi i comandi richiedono il privilegio di sistema CREATE PROCEDURE. Se il package dev'essere creato nello schema di un altro utente, e non nel proprio, si dovrà disporre del privilegio di sistema CREATE ANY PROCEDURE.

Ecco la sintassi per la creazione delle specifiche di package:

```
create [or replace] package [utente.] package
[authid {definer | current_user} ]
{is | as}
specifica package;
```

Una *specifica di package* è costituita dall'elenco delle funzioni, procedure, variabili, costanti, cursori ed eccezioni che saranno messi a disposizione degli utenti del package.

Di seguito è riportato un comando `create package` di esempio. In questo esempio, viene creato il package `GESTIONE_LIBRO`. La funzione `SPESE_RITARDO` e la procedura `NUOVO_LIBRO` presentate in precedenza sono comprese nel package.

```
create or replace package GESTIONE_LIBRO
as
    function SPESE_RITARDO(aNome IN VARCHAR2) return NUMBER;
    procedure NUOVO_LIBRO (aTitolo IN VARCHAR2,
                           aEditore IN VARCHAR2, aNomeCategoria IN VARCHAR2);
end GESTIONE_LIBRO;
```

**NOTA** Il nome dell'oggetto procedurale può essere aggiunto alla clausola `end`, come nell'esempio precedente. Questa aggiunta può rendere più agevole la coordinazione della logica all'interno del codice.

Un *corpo di package* contiene i blocchi e le specifiche per tutti gli oggetti pubblici elencati nelle specifiche del package. Il corpo potrà comprendere oggetti non elencati nelle specifiche. Tali oggetti vengono detti *privati* e non sono disponibili per gli utenti del package. Solo gli altri oggetti contenuti nel corpo dello stesso package possono chiamare gli oggetti privati. Il corpo di un package può includere anche del codice che viene eseguito ogni volta che si chiama il package, indipendentemente dalla parte del package eseguita (per un esempio, si rimanda al paragrafo “Inizializzazione di package” più avanti in questo capitolo).

La sintassi per la creazione dei corpi dei package è la seguente:

```
create [or replace] package body [utente.] corpo package
{is | as}
corpo package;
```

Il nome del corpo del package dovrebbe coincidere con quello della specifica.

Continuando con l'esempio di `GESTIONE_LIBRO`, il corpo del package può essere creato mediante il comando `create package body` riportato nel listato seguente:

```
create or replace package body GESTIONE_LIBRO
as
    function SPESE_RITARDO (aNome IN VARCHAR2)
        return NUMBER
    is
        saldo NUMBER(10,2);
    begin
        select SUM((DataRestituzione-DataPrestito) -14)*0.20
        into saldo
        from BIBLIOTECA_PRESTITO
        where Nome = aNome
            and (DataRestituzione-DataPrestito) 14;
        RETURN(saldo);
    EXCEPTION
        when NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20100,
            'No books borrowed.');
    end SPESE_RITARDO;
    procedure NUOVO_LIBRO (aTitolo IN VARCHAR2,
```

```

aEditore IN VARCHAR2, aNomeCategoria IN VARCHAR2)
is
begin
  insert into BIBLIOTECA
    (Titolo, Editore, NomeCategoria, Classificazione)
    values (aTitolo, aEditore, aNomeCategoria, NULL);
  delete from LIBRO_ORDINE
    where Titolo = aTitolo;
end NUOVO_LIBRO;
end GESTIONE_LIBRO;

```

Il comando `create or replace package body`, mostrato nell'esempio precedente, combina il comando `create function` per la funzione `SPESE_RITARDO` con il comando `create procedure` per la procedura `NUOVO_LIBRO`. Le clausole `end` hanno tutte il nome degli oggetti a esse associati e aggiunti (riportati in grassetto nel listato precedente). Questa modifica delle clausole `end` contribuisce a chiarire i punti finali del codice sorgente dell'oggetto.

All'interno del corpo del package si possono definire altre funzioni, procedure, eccezioni, variabili, cursori e costanti, che però non saranno disponibili per il pubblico, a meno che non vengano dichiarati all'interno della specifica del package (tramite il comando `create package`). Se un utente dispone del privilegio `EXECUTE` su un package, sarà in grado di accedere a tutti gli oggetti pubblici dichiarati nella specifica del package.

## Inizializzazione dei package

I package possono includere del codice da eseguire la prima volta che un utente esegue una funzione o una procedura del package nel corso di ciascuna sessione. Nell'esempio seguente, il corpo del package `GESTIONE_LIBRO` viene modificato in modo da includere un'istruzione SQL che registra il nome dell'utente corrente e l'indicatore orario per la prima volta in cui un componente del package viene eseguito all'interno della sessione. Per registrare questi valori si devono dichiarare altre due nuove variabili nel corpo del package.

Dato che le due nuove variabili sono dichiarate all'interno del corpo del package, non sono disponibili per il pubblico. All'interno del corpo del package esse sono separate dalle procedure e dalle funzioni. Il codice di inizializzazione del package è riportato in grassetto nel listato seguente:

```

create or replace package body GESTIONE_LIBRO
as
  Nome_Utente VARCHAR2(30);
  Data_Voce DATE;
  function SPESE_RITARDO (aNome IN VARCHAR2)
    return NUMBER
    is
      saldo NUMBER(10,2);
  begin
    select SUM(((DataRestituzione-DataPrestito) -14)*0.20)
    into saldo
    from BIBLIOTECA_PRESTITO
    where Nome = aNome
      and (DataRestituzione-DataPrestito) > 14;
    RETURN(saldo);
  EXCEPTION
    when NO_DATA_FOUND THEN

```

```

RAISE_APPLICATION_ERROR(-20100,
  'No books borrowed.');
end SPESE_RITARDO;
procedure NUOVO_LIBRO (aTitolo IN VARCHAR2,
  aEditore IN VARCHAR2, aNomeCategoria IN VARCHAR2)
is
begin
  insert into BIBLIOTECA
    (Titolo, Editore, NomeCategoria, Classificazione)
    values (aTitolo, aEditore, aNomeCategoria, NULL);
  delete from LIBRO_ORDINE
    where Titolo = aTitolo;
end NUOVO_LIBRO;
begin
  select Utente, SysDate
    into Nome_Utente, Data_Voce
    from DUAL;
end GESTIONE_LIBRO;

```

**NOTA** Il codice che dev'essere eseguito la prima volta che si esegue un componente del package viene memorizzato in un proprio blocco PL/SQL al termine del corpo del package. Tale codice non dispone di una propria clausola end e utilizza quella del corpo del package.

La prima volta che un componente del package GESTIONE\_LIBRO viene eseguito nella sessione dell'utente, le variabili *Nome\_Utente* e *Data\_Voce* vengono popolate dalla query mostrata nel listato precedente. Queste due variabili potranno quindi essere utilizzate dalle funzioni e dalle procedure all'interno del package, anche se in questo esempio non accade per motivi di spazio.

Per eseguire una procedura o una funzione che si trova all'interno di un package, occorre specificare il nome del package e il nome della procedura o funzione nel comando execute, come mostrato di seguito:

```
execute GESTIONE_LIBRO.NUOVO_LIBRO('SOMETHING SO STRONG', 'PANDORAS', 'SAGGIADULTI');
```

## 29.8 Visualizzazione del codice sorgente di oggetti procedurali

Il codice sorgente di procedure, funzioni, package e corpi di package già esistenti può essere sottoposto a query dalle viste del dizionario dati di seguito elencate:

---

USER_SOURCE	Per oggetti procedurali di proprietà dell'utente.
ALL_SOURCE	Per oggetti procedurali di proprietà dell'utente o per i quali all'utente è stato concesso l'accesso.
DBA_SOURCE	Per tutti gli oggetti procedurali nel database.

---

Si selezionino delle informazioni dalla vista USER\_SOURCE tramite una query simile a quella riportata nel listato seguente. In questo esempio, viene selezionata la colonna Testo, ordi-

nata per numero di Linea. Il Nome e il Tipo dell'oggetto servono per definire per quale oggetto verrà visualizzato il codice sorgente. Nell'esempio seguente, viene utilizzata la procedura NUOVO\_LIBRO descritta in precedenza nel capitolo:

```
select Testo
  from USER_SOURCE
 where Nome = 'NUOVO_LIBRO'
   and Tipo = 'PROCEDURE'
 order by Linea;

TESTO
-----
procedure NUOVO_LIBRO (aTitolo IN VARCHAR2,
  aEditore IN VARCHAR2, aNomeCategoria IN varchar2)
as
begin
  insert into BIBLIOTECA (Titolo, Editore, NomeCategoria, Classificazione)
    values (aTitolo, aEditore, aNomeCategoria, NULL);
  delete from LIBRO_ORDINE
    where Titolo = aTitolo;
end;
```

Come illustrato in questo esempio, la vista USER\_SOURCE contiene un record per ciascuna linea della procedura NUOVO\_LIBRO. La successione delle linee è mantenuta dalla colonna Linea; di conseguenza, la colonna Linea dovrebbe essere utilizzata nella clausola order by.

Valori validi per la colonna Tipo sono PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, JAVA SOURCE, TYPE e TYPE BODY.

## 29.9 Compilazione di procedure, funzioni e package

Oracle compila gli oggetti procedurali nel momento in cui vengono creati. Tuttavia, gli oggetti procedurali possono perdere validità se gli oggetti di database cui fanno riferimento cambiano. Alla successiva esecuzione, gli oggetti procedurali verranno ricompilati dal database.

Per evitare questa compilazione durante l'esecuzione e il calo di prestazioni che essa può causare, è possibile ricompilare esplicitamente le procedure, le funzioni e i package. Per ricompilare una procedura, occorre utilizzare il comando alter procedure, come mostrato nel listato seguente. La clausola compile è l'unica opzione valida per questo comando.

```
alter procedure NUOVO_LIBRO compile;
```

Per ricompilare una procedura, è necessario esserne il proprietario oppure disporre del privilegio di sistema ALTER ANY PROCEDURE.

Per ricompilare una funzione, si utilizza il comando alter function con la clausola compile:

```
alter function SPESE_RITARDO compile;
```

Per ricompilare una funzione, è necessario esserne il proprietario o disporre del privilegio di sistema ALTER ANY PROCEDURE.

Quando si ricompilano i package, si può ricompilare sia la specifica sia il corpo del package, oppure ricompilare il solo corpo. Per default, vengono ricompilati entrambi (corpo e specifica).

I comandi alter function o alter procedure non possono essere utilizzati per ricompilare funzioni e procedure memorizzate all'interno di un package.

Se il codice sorgente delle procedure o funzioni all'interno del corpo del package è cambiato, mentre la specifica del package è rimasta immutata, è possibile ricompilare solo il corpo del package. Nella maggior parte dei casi, è opportuno ricompilare entrambi.

Ecco la sintassi del comando alter package:

```
alter package [utente.] nome_package
compile [debug] [package | body | specification];
```

Per ricompilare un package, si utilizza il comando alter package precedente con la clausola compile, come mostrato di seguito:

```
alter package GESTIONE_LIBRO compile;
```

Per ricompilare un package, è necessario esserne il proprietario o disporre del privilegio di sistema ALTER ANY PROCEDURE. Dato che nell'esempio precedente non sono stati specificati né PACKAGE né BODY, è stata utilizzata l'impostazione predefinita PACKAGE, che ha comportato la ricompilazione sia della specifica sia del corpo del package.

## 29.10 Sostituzione di procedure, funzioni e package

Le procedure, le funzioni e i package possono essere sostituiti mediante i rispettivi comandi create or replace. L'uso della clausola or replace mantiene invariato qualsiasi privilegio esistente creato per gli oggetti sostituiti. Se si decide di scaricare e ricreare gli oggetti procedurali, si dovrà concedere nuovamente qualsiasi privilegio EXECUTE concesso in precedenza.

## 29.11 Eliminazione di procedure, funzioni e package

Per eliminare una procedura, si utilizza il comando drop procedure, come mostrato di seguito:

```
drop procedure NUOVO_LIBRO;
```

Per eliminare una procedura, è necessario esserne il proprietario oppure possedere il privilegio di sistema DROP ANY PROCEDURE.

Per eliminare una funzione, si utilizza il comando drop function, come mostrato di seguito:

```
drop function SPESE_RITARDO;
```

Per eliminare una funzione, è necessario esserne il proprietario o disporre del privilegio di sistema DROP ANY PROCEDURE.

Per eliminare un package (specifiche e corpo), si utilizza il comando drop package, come mostrato di seguito:

```
drop package GESTIONE_LIBRO;
```

Per eliminare un package, è necessario esserne il proprietario o possedere il privilegio di sistema DROP ANY PROCEDURE.

Per eliminare il corpo di un package, si utilizza il comando drop package con la clausola body, come nell'esempio seguente:

```
drop package body GESTIONE_LIBRO;
```

Per eliminare il corpo di un package, è necessario esserne il proprietario o possedere il privilegio di sistema DROP ANY PROCEDURE.



Parte quarta

**DATABASE RELAZIONALI  
A OGGETTI**



## Capitolo 30

# Implementazione di tipi, viste oggetto e metodi

- 30.1 **Nuova analisi dei tipi di dati astratti**
- 30.2 **Implementazione di viste oggetto**
- 30.3 **Metodi**

In questo capitolo, sono riportate ulteriori informazioni sull'uso dei tipi di dati astratti, presentati per la prima volta nel Capitolo 4. Il capitolo descrive argomenti come l'amministrazione della sicurezza per i tipi di dati astratti e l'indicizzazione degli attributi di questi tipi di dati. Viene descritta anche la creazione di metodi per i tipi di dati astratti, insieme all'impiego di viste oggetto e di trigger di tipo INSTEAD OF.

Per usare le informazioni fornite in questo capitolo, si dovrebbe avere già una certa familiarità con i tipi di dati astratti (Capitolo 4), le viste (Capitolo 18), il comando grant (Capitolo 19), gli indici (Capitolo 20), le procedure e i trigger PL/SQL (Capitoli 27, 28 e 29). Gli elementi di ciascuno di questi argomenti ricoprono un ruolo sostanziale nell'implementazione di un database relazionale a oggetti. Il primo paragrafo del capitolo è dedicato a una seconda analisi dei tipi di dati astratti descritti nel Capitolo 4.

## 30.1 Nuova analisi dei tipi di dati astratti

Come descritto nel Capitolo 4, i tipi di dati astratti possono essere utilizzati per raggruppare colonne correlate in oggetti. Per esempio, le colonne che fanno parte di informazioni relative a indirizzi possono essere raggruppate in un tipo di dati INDIRIZZO\_TY tramite il comando create type:

```
create type INDIRIZZO_TY as object
(Via      VARCHAR2(50),
Citta    VARCHAR2(25),
Prov     CHAR(2),
Cap      NUMBER);
```

**NOTA** *In base allo strumento impiegato, per eseguire questo comando potrebbe essere necessario aggiungere un simbolo “/” alla fine.*

Il comando create type crea nel listato precedente un tipo di dati astratto INDIRIZZO\_TY, che può essere utilizzato quando si creano altri oggetti di database. Per esempio, il comando create type seguente crea il tipo di dati PERSONA\_TY utilizzando il tipo di dati INDIRIZZO\_TY come tipo di dati per la colonna Indirizzo:

```
create type PERSONA_TY as object
(Nome      VARCHAR2(25),
Indirizzo INDIRIZZO_TY);
```

Dato che la colonna Indirizzo del tipo di dati PERSONA\_TY utilizza il tipo di dati INDIRIZZO\_TY, non contiene un solo valore, bensì quattro, ossia le quattro colonne che costituiscono il tipo di dati INDIRIZZO\_TY.

## Sicurezza per i tipi di dati astratti

Nell'esempio precedente si assumeva che lo stesso utente possedesse sia il tipo di dati INDIRIZZO\_TY, sia il tipo di dati PERSONA\_TY. Cosa sarebbe accaduto se il proprietario del tipo di dati PERSONA\_TY non avesse posseduto anche il tipo di dati INDIRIZZO\_TY?

Per esempio, cosa accade se l'account "Dora" possiede il tipo di dati INDIRIZZO\_TY e l'utente dell'account "George" cerca di creare il tipo di dati PERSONA\_TY? George eseguirà il comando riportato di seguito:

```
create type PERSONA_TY as object
(Nome      VARCHAR2(25),
Indirizzo INDIRIZZO_TY);
```

Se George non è il proprietario del tipo di dati astratto INDIRIZZO\_TY, Oracle risponderà a questo comando `create type` con il messaggio seguente:

Warning: Type created with compilation errors.

Gli errori di compilazione sono causati da problemi nella creazione del *constructor method*, un metodo speciale creato da Oracle al momento della creazione del tipo di dati. Oracle non è in grado di risolvere il riferimento al tipo di dati INDIRIZZO\_TY, in quanto George non possiede un tipo di dati con quel nome. George potrebbe eseguire nuovamente il comando `create type` (con la clausola `or replace`) per fare un riferimento specifico al tipo di dati INDIRIZZO\_TY di Dora:

```
create or replace type PERSONA_TY as object
(Nome      VARCHAR2(25),
Indirizzo Dora.INDIRIZZO_TY);
```

Warning: Type created with compilation errors.

Per visualizzare gli errori associati alla creazione del tipo di dati, si utilizza il comando `show errors`:

```
show errors
Errors for TYPE PERSONA_TY:
LINE/COL  ERROR
----- 
0/0      PL/SQL: Compilation unit analysis terminated
3/11    PLS-00201: identifier 'DORA.INDIRIZZO_TY' must be declared
```

George non sarà in grado di creare il tipo di dati PERSONA\_TY (che comprende il tipo di dati INDIRIZZO\_TY), a meno che Dora non gli conceda il privilegio EXECUTE sul proprio tipo di dati. Il listato seguente mostra la concessione di tale privilegio:

```
grant EXECUTE on INDIRIZZO_TY to George;
```

Ottenuti i privilegi richiesti, George potrà creare un tipo di dati basato sul tipo di dati INDIRIZZO\_TY di Dora, come mostrato nel listato seguente.

```
create or replace type PERSONA_TY as object
(Nome      VARCHAR2(25),
Indirizzo  Dora.INDIRIZZO_TY);
```

L'uso dei tipi di dati di un altro utente è significativo. Per esempio, durante le operazioni di inserimento è necessario specificare il nome del proprietario di ciascun tipo di dati. George sarà in grado di creare una tabella basata sul proprio tipo di dati PERSONA\_TY (che include il tipo di dati INDIRIZZO\_TY di Dora), come nell'esempio seguente:

```
create table CLIENTE
(Cliente_ID NUMBER,
Persona      PERSONA_TY);
```

Se George possedesse i tipi di dati PERSONA\_TY e INDIRIZZO\_TY, un'operazione di inserimento nella tabella CLIENTE utilizzerebbe il formato seguente:

```
insert into CLIENTE values
(1,PERSONA_TY('QualcheNome',
INDIRIZZO_TY('ValoreVia','ValoreCitta','ST',11111)));
```

Dato che George non possiede il tipo di dati INDIRIZZO\_TY, questo comando non viene completato con successo. Durante l'operazione di inserimento viene utilizzato il constructor method INDIRIZZO\_TY, posseduto da Dora. Pertanto il comando insert dev'essere modificato in modo da specificare Dora come proprietario del tipo di dati INDIRIZZO\_TY. Nell'esempio seguente è descritta l'istruzione insert corretta, con il riferimento a Dora evidenziato in grassetto:

```
insert into CLIENTE values
(1,PERSONA_TY('QualcheNome',
Dora.INDIRIZZO_TY('ValoreVia','ValoreCitta','ST',11111)));
```

George può usare un sinonimo per il tipo di dati di Dora? No. Anche se *può* creare un sinonimo denominato INDIRIZZO\_TY:

```
create synonym INDIRIZZO_TY for Dora.INDIRIZZO_TY;
```

questo sinonimo non può essere utilizzato:

```
create type PERSONA2_TY
(Nome      VARCHAR2(25),
Indirizzo  INDIRIZZO_TY); Type created with compilation errors.
```

Non si può utilizzare un sinonimo per il tipo di dati di un altro utente. Pertanto si dovrà fare riferimento al proprietario del tipo di dati durante ciascun comando insert.

**NOTA** Quando si crea un sinonimo, Oracle non verifica la validità dell'oggetto per il quale si crea il sinonimo. Quando si impara il comando create synonym x for y, Oracle non verifica che "y" sia un nome di oggetto valido o un tipo di oggetto valido. La validità di questo oggetto viene verificata solo quando il sinonimo vi accede.

È opportuno accertarsi di aver eliminato il sinonimo prima di continuare con gli esempi:

```
drop synonym INIDIRIZZO_TY;
```

In un'implementazione esclusivamente relazionale di Oracle, il privilegio EXECUTE viene concesso su oggetti procedurali, come procedure e package. Nell'ambito dell'implementazione relazionale a oggetti di Oracle, il privilegio EXECUTE viene esteso a coprire anche i tipi di dati astratti. Questo privilegio viene utilizzato perché i tipi di dati astratti possono comprendere *metodi*, funzioni e procedure PL/SQL che operano sul tipo di dati. Se si concede a terzi il privilegio di utilizzare il proprio tipo di dati, si concede a tali utenti il privilegio di eseguire i metodi definiti sul tipo di dati. Anche se Dora non ha ancora definito dei metodi sul tipo di dati INDIRIZZO\_TY, Oracle crea automaticamente dei constructor method per accedere ai dati. Qualsiasi oggetto (come PERSONA\_TY) che usi il tipo di dati INDIRIZZO\_TY utilizzerà il constructor method associato a INDIRIZZO\_TY. Anche se non si è creato alcun metodo per il tipo di dati astratto, esistono comunque delle procedure associate a quest'ultimo.

Si può descrivere CLIENTE:

```
describe CLIENTE
```

Nome	Null?	Type
CLIENTE_ID		NUMBER
PERSONA		NAMED TYPE

Per mostrare gli attributi del tipo di dati per le tabelle che utilizzano caratteristiche relazionali a oggetti, si può ricorrere al comando set describe depth in SQL\*Plus. Si possono specificare valori di profondità compresi tra 1 e 50:

```
set describe depth 2
```

```
desc CLIENTE
```

Nome	Null?	Type
CLIENTE_ID		NUMBER
PERSONA		PERSONA_TY
NOME		VARCHAR2(25)
INDIRIZZO		DORA.INDIRIZZO_TY

```
set describe depth 3
```

```
desc cliente
```

Nome	Null?	Type
CLIENTE_ID		NUMBER
PERSONA		PERSONA_TY
NOME		VARCHAR2(25)
INDIRIZZO		DORA.INDIRIZZO_TY
VIA		VARCHAR2(50)
CITTA		VARCHAR2(25)
PROVINCIA		CHAR(2)
CAP		NUMBER

## Indicizzazione degli attributi di un tipo di dati astratto

Nell'esempio del Capitolo 4 veniva creata la tabella CLIENTE, la quale, come illustrato nel listato seguente, contiene una colonna normale, Cliente\_ID, e una colonna Persona definita dal tipo di dati astratto PERSONA\_TY:

```
create table CLIENTE
(Cliente_ID NUMBER,
Persona    PERSONA_TY);
```

Dalle definizioni del tipo di dati riportate nel paragrafo precedente del capitolo si può evincere che PERSONA\_TY dispone di una colonna Nome, seguita da una colonna Indirizzo definita dal tipo di dati INDIRIZZO\_TY.

Come mostrato nel Capitolo 4, gli attributi del tipo di dati vengono completamente qualificati quando si fa riferimento alle colonne all'interno dei tipi di dati astratti, anteponendo al nome dell'attributo il nome della colonna, e facendo precedere quest'ultimo da una variabile di correlazione. Per esempio, la query seguente restituisce la colonna Cliente\_ID insieme alla colonna Nome. La colonna Nome è un attributo del tipo di dati che definisce la colonna Persona, per cui si dovrà fare riferimento all'attributo con la notazione Persona.Nome:

```
select Cliente_ID, C.Persona.Nome
  from CLIENTE C;
```

**NOTA** Quando si esegue una query sugli attributi di un tipo di dati astratto, è necessario utilizzare una variabile di correlazione per la tabella, come mostrato in questo esempio.

Per impostare gli attributi di visualizzazione per gli attributi di un tipo di dati astratto in SQL\*Plus, si utilizza il comando column, specificando il nome del tipo di dati e l'attributo:

```
column Cliente_ID format 9999999
column Persona.Nome format A25
```

Per fare riferimento ad attributi all'interno del tipo di dati INDIRIZZO\_TY, occorre specificare il percorso completo attraverso le colonne correlate. Per esempio, alla colonna Via si fa riferimento tramite Persona.Indirizzo.Via, descrivendo per esteso la sua collocazione all'interno della struttura della tabella. Nell'esempio seguente, si fa riferimento due volte alla colonna Citta: nell'elenco di colonne da selezionare e all'interno della clausola where. In ogni riferimento, è richiesta una variabile di correlazione (in questo caso, C ).

```
select C.Persona.Nome,
       C.Persona.Indirizzo.Citta
  from CLIENTE C
 where C.Persona.Indirizzo.Citta like 'F%';
```

Dato che la colonna Citta viene utilizzata con un intervallo di ricerca all'interno della clausola where, l'ottimizzatore di Oracle è in grado di utilizzare un indice al momento di risolvere la query. Se nella colonna Citta è disponibile un indice, Oracle potrà individuare rapidamente tutte le righe i cui valori di Citta iniziano con la lettera 'F', come richiesto dalla query.

**NOTA** Il funzionamento interno dell'ottimizzatore di Oracle, e le condizioni alle quali vengono utilizzati gli indici, sono descritti nel Capitolo 38. Per le linee guida riguardanti le colonne da indicizzare si rimanda sempre a questo capitolo.

Per creare un indice su una colonna che faccia parte di un tipo di dati astratto, si deve specificare il percorso completo sino alla colonna come parte del comando `create index`. Per creare un indice sulla colonna Citta (che fa parte della colonna Indirizzo), è possibile eseguire il comando riportato di seguito:

```
create index I_CLIENTE$CITTA
on CLIENTE(Persona.Indirizzo.Citta);
```

Questo comando creerà un indice di nome `I_CLIENTE$CITTA` sulla colonna `Persona.Indirizzo.Citta`. Ogni volta che si accede alla colonna Citta, l'ottimizzatore di Oracle valuterà il codice SQL utilizzato per accedere ai dati e stabilirà se il nuovo indice può contribuire a migliorare le prestazioni della query.

**NOTA** *Nel comando `create index` non si ricorre alle variabili di correlazione quando si fa riferimento agli attributi di tipi di dati astratti.*

Per creare tabelle basate su tipi di dati astratti, è necessario considerare come avviene l'accesso alle colonne all'interno di tali tipi di dati. Se, come avviene per la colonna Citta nell'esempio precedente, alcune colonne vengono utilizzate normalmente come parte delle condizioni di limitazione nelle query, queste colonne devono essere indicizzate. A questo riguardo, la rappresentazione di più colonne in un unico tipo di dati astratto può determinare un peggioramento delle prestazioni dell'applicazione, in quanto può mascherare l'esigenza di indicizzare colonne specifiche all'interno del tipo di dati. Con i tipi di dati astratti, ci si abitua a trattare un gruppo di colonne come un'unica entità, come le colonne Indirizzo o le colonne Persona. È importante ricordare che l'ottimizzatore, in sede di valutazione dei percorsi di accesso della query, prende in considerazione le colonne singolarmente. Inoltre, l'indicizzazione della colonna Citta in una tabella che utilizza il tipo di dati `INDIRIZZO_TY` non influisce sulla colonna Citta presente in una seconda tabella che usa il tipo di dati `INDIRIZZO_TY`. Per esempio, se esiste una seconda tabella di nome `FILIALE` che utilizza il tipo di dati `INDIRIZZO_TY`, la sua colonna Citta non verrà indicizzata, a meno che non si crei un indice per essa. La presenza di un indice sulla colonna Citta nella tabella `CLIENTE` non influisce sulla colonna Citta nella tabella `FILIALE`.

Pertanto, uno dei vantaggi offerti dai tipi di dati astratti, la capacità di migliorare l'aderenza agli standard per la rappresentazione di dati logici, non si estende alla rappresentazione dei dati fisici. Ciascuna implementazione fisica dev'essere curata separatamente.

## 30.2 Implementazione di viste oggetto

La tabella `CLIENTE` creata nel paragrafo precedente presupponeva l'esistenza di un tipo di dati `PERSONA_TY`. Per l'implementazione di applicazioni che richiedono database relazionali a oggetti, per prima cosa si devono utilizzare i metodi di progettazione per database relazionali descritti nella prima parte del libro. Una volta che il progetto del database è adeguatamente normalizzato, è necessario individuare gruppi di colonne (o colonne singole) che costituiscano la rappresentazione di un tipo di dati astratto. Sarà quindi possibile creare tabelle basate sui tipi di dati astratti, come descritto in precedenza in questo capitolo.

Come ci si comporta se le tabelle esistono già? Come si procede se in precedenza è già stata creata un'applicazione di database relazionale e ora si desidera implementare concetti relazionali a oggetti nella medesima applicazione senza doverla ricostruire interamente? In questo caso, è necessario riuscire a sovrapporre strutture orientate agli oggetti (OO, Object-Oriented), come i

tipi di dati astratti, su tabelle relazionali già esistenti. Oracle mette a disposizione delle *viste oggetto* per la definizione di oggetti utilizzati da tabelle relazionali già esistenti.

**NOTA** *In tutti gli esempi del libro, i nomi delle viste oggetto terminano sempre con il suffisso "OV".*

Negli esempi riportati in precedenza nel capitolo, l'ordine delle operazioni era il seguente:

1. Creazione del tipo di dati INDIRIZZO\_TY.
2. Creazione del tipo di dati PERSONA\_TY, utilizzando il tipo di dati INDIRIZZO\_TY.
3. Creazione della tabella CLIENTE, utilizzando il tipo di dati PERSONA\_TY.

Se la tabella CLIENTE esiste già, si possono creare i tipi di dati INDIRIZZO\_TY e PERSONA\_TY, quindi ricorrere alle viste oggetto per correlare questi tipi di dati alla tabella CLIENTE. Nel listato seguente, la tabella CLIENTE viene creata come una tabella relazionale, utilizzando solo i tipi di dati incorporati:

```
drop table CLIENTE;

create table CLIENTE
(Cliente_ID NUMBER primary key,
Nome      VARCHAR2(25),
Via       VARCHAR2(50),
Citta     VARCHAR2(25),
Prov      CHAR(2),
Cap       NUMBER);
```

Per creare un'altra tabella o applicazione che memorizzi informazioni su persone e indirizzi, si può scegliere di creare i tipi di dati INDIRIZZO\_TY e PERSONA\_TY. Per coerenza, questi dovrebbero essere applicati anche alla tabella CLIENTE.

Con la tabella CLIENTE appena creata (e possibilmente popolata di dati), è possibile creare tipi di dati astratti. In primo luogo, si crei INDIRIZZO\_TY:

```
drop type PERSONA_TY;
drop type INDIRIZZO_TY;
create or replace type INDIRIZZO_TY as object
(Via      VARCHAR2(50),
Citta    VARCHAR2(25),
Prov      CHAR(2),
Cap       NUMBER);
```

A questo punto, si crei PERSONA\_TY, che utilizza INDIRIZZO\_TY:

```
create or replace type PERSONA_TY as object
(Nome      VARCHAR2(25),
Indirizzo  INDIRIZZO_TY);
```

Ora si crei una vista oggetto basata sulla tabella CLIENTE, utilizzando i tipi di dati definiti. Una vista oggetto inizia con il comando *create view*. All'interno del comando *create view* è possibile specificare la query che costituirà la base della vista. Verranno utilizzati i tipi di dati appena creati. Il codice necessario per la creazione della vista oggetto CLIENTE\_OV è riportato nel listato seguente:

```
create view CLIENTE_OV (Cliente_ID, Persona) as
select Cliente_ID,
       PERSONA_TY(Nome,
       INDIRIZZO_TY(Via, Citta, Prov, Cap))
from CLIENTE;
```

Si analizzi in dettaglio questo comando. Nella prima linea, si assegna un nome alla vista oggetto:

```
create view CLIENTE_OV (Cliente_ID, Persona) as
```

La vista CLIENTE\_OV è composta da due colonne: Cliente\_ID e Persona (quest'ultima è definita dal tipo di dati PERSONA\_TY). Si osservi che non è possibile specificare “object” come opzione all'interno del comando `create view`.

Nella prossima sezione del comando `create view`, viene creata la query che formerà la base della vista:

```
select Cliente_ID,
       PERSONA_TY(Nome,
       INDIRIZZO_TY(Via, Citta, Prov, Cap))
from CLIENTE;
```

Questo esempio mostra alcuni impieghi interessanti della sintassi. Quando si crea una tabella su tipi di dati astratti già esistenti, i valori delle colonne vengono selezionati dalla tabella facendo riferimento ai nomi delle colonne (come Persona e Indirizzo, negli esempi precedenti) invece che ai loro constructor method. Per la creazione di una vista oggetto, occorre fare riferimento ai nomi dei constructor method (PERSONA\_TY e INDIRIZZO\_TY).

**NOTA** *Per esempi di query che fanno riferimento a colonne definite da tipi di dati astratti, si rimanda al Capitolo 4.*

Nella query che costituisce la base della vista oggetto è possibile utilizzare una clausola `where`. Nell'esempio seguente, CLIENTE\_OV viene modificata in modo da includere una clausola `where` che la limita alla sola visualizzazione dei valori CLIENTE per i quali la colonna Prov contiene un valore ‘MI’:

```
create or replace view CLIENTE_OV (Cliente_ID, Persona) as
select Cliente_ID,
       PERSONA_TY(Nome,
       INDIRIZZO_TY(Via, Citta, Prov, Cap))
from CLIENTE
where Prov = 'MI';
```

Quando si crea la vista oggetto CLIENTE\_OV, la clausola `where` della query di base della vista non fa riferimento al tipo di dati astratto. Al contrario, essa fa riferimento direttamente alla tabella CLIENTE. La clausola `where` fa riferimento direttamente alla colonna Prov perché la query si riferisce direttamente alla tabella CLIENTE, una tabella relazionale che non si basa su tipi di dati astratti.

Per creare viste oggetto basate su tabelle relazionali, l'ordine delle operazioni è il seguente:

1. Creazione della tabella CLIENTE se non esiste ancora.
2. Creazione del tipo di dati INDIRIZZO\_TY.

3. Creazione del tipo di dati PERSONA\_TY, utilizzando il tipo di dati INDIRIZZO\_TY.
4. Creazione della vista oggetto CLIENTE\_OV, utilizzando i tipi di dati definiti.

L'uso di viste oggetto presenta due vantaggi principali. Innanzitutto, queste consentono di creare tipi di dati astratti che operano su tabelle già esistenti; dato che si possono utilizzare gli stessi tipi di dati astratti in più tabelle dell'applicazione, si potrà migliorare l'aderenza dell'applicazione alla rappresentazione standard dei dati e la capacità di riutilizzare gli oggetti già esistenti. Inoltre, dal momento che si possono definire metodi per i tipi di dati astratti, questi metodi verranno applicati sia ai dati contenuti nelle nuove tabelle, sia ai dati delle tabelle preesistenti. In secondo luogo, le viste oggetto offrono due modi diversi per inserire i dati nella tabella base. La flessibilità nella manipolazione di dati per le viste oggetto, in grado di trattare la tabella base sia come una tabella relazionale, sia come una tabella oggetto, rappresenta un vantaggio significativo per gli sviluppatori di applicazioni. Nel paragrafo seguente, vengono descritte le opzioni per la manipolazione dei dati disponibili per le viste oggetto.

## Manipolazione di dati mediante viste oggetto

I dati nella tabella CLIENTE possono essere aggiornati tramite la vista oggetto CLIENTE\_OV, oppure aggiornando direttamente la tabella CLIENTE. Trattando CLIENTE come una semplice tabella, i dati potranno essere inseriti in questa tabella con un normale comando SQL insert, come mostrato nell'esempio seguente:

```
insert into CLIENTE values
(123, 'SIGMUND', '47 HAFFNER RD', 'LEWISTON', 'NJ', 22222);
```

Questo comando insert inserisce un singolo record nella tabella CLIENTE. Anche se sulla tabella è stata creata una vista oggetto, è sempre possibile considerare la tabella CLIENTE come una tabella relazionale convenzionale.

Dato che la vista oggetto è stata creata sulla tabella CLIENTE, è possibile eseguire operazioni di inserimento in CLIENTE tramite i constructor method utilizzati dalla vista. Come mostrato in precedenza in questo capitolo, quando si inseriscono dei dati in un oggetto che utilizza tipi di dati astratti, è necessario specificare i nomi dei constructor method all'interno del comando insert. L'esempio proposto nel listato seguente descrive l'inserimento di un singolo record nella vista oggetto CLIENTE\_OV, utilizzando i constructor method CLIENTE\_TY, PERSONA\_TY e INDIRIZZO\_TY:

```
insert into CLIENTE_OV values
(234,
PERSONA_TY('EVELYN',
INDIRIZZO_TY('555 HIGH ST', 'LOWLANDS PARK', 'NE', 33333)));
```

Dato che si possono utilizzare entrambi i metodi per inserire i valori nella vista oggetto CLIENTE\_OV, è possibile standardizzare il modo in cui l'applicazione manipola i dati. Se le operazioni di inserimento si basano tutte su tipi di dati astratti, per queste operazioni si potrà utilizzare lo stesso tipo di codice, indipendentemente dal fatto che i tipi di dati astratti siano stati creati prima o dopo la tabella.

È possibile eseguire una query sulle righe appena inserite da CLIENTE o CLIENTE\_OV; ma dato che la vista contiene una clausola where, CLIENTE\_OV mostrerà solamente quelle righe che passano questa condizione di limitazione.

## Uso dei trigger INSTEAD OF

Per la creazione di una vista oggetto è possibile utilizzare un trigger INSTEAD OF. Le viste non possono utilizzare la tabella standard basata sui trigger, descritta nel Capitolo 28. I trigger INSTEAD OF possono essere utilizzati per indicare a Oracle come aggiornare le tabelle di base che fanno parte della vista. Questi trigger si applicano sia a viste oggetto, sia a viste relazionali convenzionali.

Per esempio, se una vista comporta un join di due tabelle, la possibilità di aggiornare i record nella vista è limitata. Tuttavia, se si usa un trigger INSTEAD OF, è possibile specificare a Oracle come aggiornare, cancellare o inserire i record nelle tabelle nel caso in cui un utente cerchi di modificare i valori tramite la vista. Il codice contenuto nel trigger INSTEAD OF viene eseguito *al posto del* comando insert, update o delete inserito.

Per esempio, si può avere una vista che unisce tramite join la tabella BIBLIOTECA alla tabella BIBLIOTECA\_AUTORE:

```
create or replace view AUTORE_EDITORE as
select BA.NomeAutore, Titolo, B.Editore
  from BIBLIOTECA_AUTORE BA inner join BIBLIOTECA B
     using (Titolo);
```

È possibile selezionare dei valori da questa vista e, con i trigger INSTEAD OF, manipolare i dati mediante la vista stessa. Si considerino questi record:

```
select NomeAutore, Editore from AUTORE_EDITORE
  where Nome_Autore = 'W. P. KINSELLA';
```

NOMEAUTORE	EDITORE
W. P. KINSELLA	MARINER
W. P. KINSELLA	BALLANTINE

Se si cerca di aggiornare il valore di Editore, questo tentativo non riesce:

```
update AUTORE_EDITORE
   set Editore = 'MARINER'
 where NomeAutore = 'W. P. KINSELLA';

set Editore = 'MARINER'
 *
ERROR at line 2:
ORA-01720: cannot modify a column which maps to a
           non key-preserved table
```

Il problema è che Oracle non è in grado di stabilire quale editore di quale record aggiornare in BIBLIOTECA. Per eseguire l'aggiornamento tramite la vista, si dovrà adoperare un trigger INSTEAD OF.

Nel listato seguente viene creato un trigger INSTEAD OF per la vista AUTORE\_EDITORE:

```
create trigger AUTHOR_PUBLISHER_UPDATE
instead of UPDATE on AUTORE_EDITORE
for each row
begin
  if :vecchio.Editore <> :nuovo.Editore
```

```

then
  update BIBLIOTECA
    set Editore = :nuovo.Editore
      where Titolo = :vecchio.Titolo;
end if;
if :vecchio.NomeAutore <> :nuovo.NomeAutore
then
  update BIBLIOTECA_AUTORE
    set NomeAutore = :nuovo.NomeEditore
      where Titolo = :vecchio.NomeAutore;
end if;
end;

```

Nella prima parte di questo trigger, si assegna un nome al trigger stesso e se ne descrive l'uso tramite la clausola instead of. Il nome assegnato al trigger ne descrive il funzionamento: il trigger serve per supportare i comandi update eseguiti sulla vista AUTORE\_EDITORE. Si tratta di un trigger a livello di riga; ogni riga modificata viene elaborata, come mostrato di seguito.

```

create trigger AUTORE_EDITORE_UPDATE
instead of UPDATE on AUTORE_EDITORE
for each row

```

La sezione successiva del trigger specifica a Oracle come eseguire l'aggiornamento. Durante il primo controllo all'interno del corpo del trigger, viene verificato il valore Editore. Se il vecchio valore Editore è uguale a quello nuovo, non vengono apportate modifiche. Se i valori non sono identici, la tabella BIBLIOTECA viene aggiornata con il nuovo valore Editore:

```

begin
  if :vecchio.Editore <> :nuovo.Editore
  then
    update BIBLIOTECA
      set Editore = :nuovo.Editore
        where Titolo = :vecchio.Titolo;
  end if;

```

Nella sezione successiva del trigger viene valutato il valore NomeAutore. Se questo valore è cambiato, la tabella BIBLIOTECA viene aggiornata in modo da contenere il nuovo valore NomeAutore:

```

if :vecchio.NomeAutore <> :nuovo.NomeAutore
then
  update BIBLIOTECA_AUTORE
    set NomeAutore = :nuovo.NomeAutore
      where Titolo = :vecchio.Titolo;
end if;

```

Pertanto, la vista si basa su due tavole, BIBLIOTECA e BIBLIOTECA\_AUTORE, e un aggiornamento della vista consente di aggiornare *nessuna o entrambe* le tavole. Il trigger INSTEAD OF, ideato per supportare le viste oggetto, rappresenta uno strumento potente per lo sviluppo di applicazioni.

Ora è possibile aggiornare la vista AUTORE\_EDITORE direttamente e fare in modo che il trigger aggiorni la tabella di base appropriata. Per esempio, il comando seguente consente di aggiornare la tabella BIBLIOTECA:

```
update AUTORE_EDITORE
  set Editore = 'MARINER'
 where NomeAutore = 'W. P. KINSELLA';

2 rows updated.
```

L'avvenuto aggiornamento può essere verificato con una query sulla tabella BIBLIOTECA:

```
select Editore from BIBLIOTECA
  where Titolo in
(select Titolo from BIBLIOTECA_AUTORE
 where NomeAutore = 'W. P. KINSELLA');

EDITORE
-----
MARINER
MARINER
```

I trigger INSTEAD OF sono molto potenti. Come illustrato in questo esempio, possono essere utilizzati per eseguire operazioni su tabelle differenti del database, utilizzando la logica di controllo del flusso disponibile all'interno del linguaggio PL/SQL. Quando si gestiscono le viste oggetto, si possono utilizzare i trigger INSTEAD OF per reindirizzare DML dalle viste oggetto alle tabelle base delle viste.

### 30.3 Metodi

Per la creazione di una vista oggetto sulla tabella CLIENTE sono stati definiti tipi di dati astratti utilizzati dalla tabella stessa. Questi tipi di dati aiutano a standardizzare la rappresentazione dei dati, ma hanno anche un altro scopo. È possibile definire metodi che si applichino ai tipi di dati, e applicando questi ultimi a tabelle già esistenti sarà possibile applicare i metodi ai dati contenuti nelle tabelle stesse.

Si consideri la vista oggetto CLIENTE\_OV:

```
create or replace view CLIENTE_OV (Cliente_ID, Persona) as
select Cliente_ID,
       PERSONA_TY(Nome,
      INDIRIZZO_TY(Via, Citta, Prov, Cap))
  from CLIENTE;
```

Questa vista applica i tipi di dati astratti PERSONA\_TY e INDIRIZZO\_TY ai dati contenuti nella tabella CLIENTE. Se questi tipi di dati sono associati a dei metodi, questi potranno essere applicati ai dati della tabella. Per la creazione di un tipo di dati astratto si può utilizzare il comando `create type`. Per la creazione di un metodo si utilizza il comando `create type body`.

#### Sintassi per la creazione dei metodi

Prima di creare il corpo di un metodo, occorre assegnare a quest'ultimo un nome all'interno della dichiarazione del tipo di dati. Per esempio, si crei un nuovo tipo di dati, ANIMALE\_TY. La definizione di ANIMALE\_TY è riportata nel listato seguente:

```
create or replace type ANIMALE_TY as object
(Generazione VARCHAR2(25),
Nome VARCHAR2(25),
DataNascita DATE,
member function ETA (DataNascita IN DATE) return NUMBER);
```

Il tipo di dati ANIMALE\_TY può essere utilizzato come parte di una vista oggetto sulla tabella ALLEVAMENTO utilizzata nel Capitolo 13. In questa tabella sono registrati il nome di un discendente, il suo sesso, i suoi genitori, e la sua data di nascita.

All'interno del comando `create type`, la linea:

```
member function ETA (DataNascita IN DATE) return NUMBER)
```

assegna un nome alla funzione che rappresenta un “membro” del tipo di dati ANIMALE\_TY. Dato che ETA restituisce un valore, si tratta di una funzione (si consulti il Capitolo 29). Per definire la funzione ETA, occorre utilizzare il comando `create type body`, la cui sintassi completa è descritta nel Capitolo 42.

Si crei la funzione ETA come funzione membro all'interno del tipo di dati ANIMALE\_TY. La funzione ETA restituisce l'età degli animali in giorni:

```
create or replace type body ANIMALE_TY as
member function Eta (DataNascita DATE) return NUMBER is
begin
    RETURN ROUND(SysDate - DataNascita);
end;
end;
```

Per codificare altre funzioni o procedure, è necessario specificarle all'interno dello stesso comando `create type body`, prima della clausola finale `end`.

Completata la creazione della funzione ETA, questa viene associata al tipo di dati ANIMALE\_TY. Se una tabella utilizza questo tipo di dati, sarà in grado di utilizzare la funzione ETA. Esiste però un problema nel modo in cui la funzione viene specificata. La funzione ETA, così come definita precedentemente nel comando `create type body`, non applica restrizioni esplicite agli aggiornamenti. Questa funzione non aggiorna alcun dato, ma non si hanno garanzie che il corpo del tipo non venga modificato in futuro per fare in modo che la funzione ETA aggiorni dei dati. Per chiamare una funzione membro all'interno di un'istruzione SQL, è necessario assicurarsi che la funzione non sia in grado di aggiornare il database. Per questo motivo, si dovrà modificare la specifica della funzione nel comando `create type`. La modifica è riportata in grassetto nel listato seguente:

```
create or replace type ANIMALE_TY as object
(Generazione VARCHAR2(25),
Nome VARCHAR2(25),
DataNascita DATE,
member function ETA (DataNascita IN DATE) return NUMBER,
PRAGMA RESTRICT_REFERENCES(ETA, WNDS);
```

**NOTA** Non è possibile eliminare o ricreare un tipo attualmente utilizzato in una tabella. Di conseguenza, è molto importante apportare questa modifica prima di creare una tabella che usa il tipo di dati ANIMALE\_TY.

La linea seguente:

```
PRAGMA RESTRICT_REFERENCES(ETA, WNDS));
```

indica a Oracle che la funzione ETA opera nello stato Write No Database State, quindi non è in grado di modificare il database. Altre restrizioni disponibili sono Read No Database State (RNDS, non consente l'esecuzione di query), Write No Package State (WNPS, non consente la modifica di valori di variabili all'interno di package) e Read No Package State (RNPS, non consente di fare riferimento a valori di variabili all'interno di package).

La funzione ETA può ora essere utilizzata all'interno di una query. Se ANIMALE\_TY viene utilizzato come tipo di dati per una colonna di nome Animale nella tabella ANIMALE, la struttura della tabella può essere quella riportata di seguito:

```
create table ANIMALE
(ID      NUMBER,
Animale  ANIMALE_TY);
insert into ANIMALE values
(11,ANIMALE_TY('MUCCA','MIMI',TO_DATE('01-JAN-1997','DD-MON-YYYY')));
```

Ora è possibile invocare la funzione ETA, che fa parte del tipo di dati ANIMALE\_TY. Come accade con gli attributi di un tipo di dati astratto, è possibile fare riferimento ai metodi membro tramite i nomi delle colonne che utilizzano i tipi di dati (in questo caso, Animale.ETA):

```
select A.Animale.ETA(A.Animale.DataNascita)
  from ANIMALE A;
A.ANIMALE.ETA(A.ANIMALE.DATANASCITA)
-----
1987
```

La chiamata ad A.Animale.ETA(A.Animale.DataNascita) esegue il metodo ETA all'interno del tipo di dati ANIMALE\_TY. Di conseguenza, non è necessario memorizzare i dati di variabili come l'età all'interno del database. Al contrario, si possono memorizzare nel database dati statici, come la data di nascita, e utilizzare le chiamate a funzioni e procedure per derivare i dati della variabile.

**NOTA** *In questo esempio, è necessario utilizzare le variabili di correlazione in due punti: quando si chiama il metodo ETA e quando si passa il nome della colonna Datanascita a questo metodo.*

## Gestione dei metodi

È possibile aggiungere nuovi metodi a un tipo di dati modificando il tipo di dati stesso (mediante il comando alter type). Per modificare un tipo di dati occorre elencare tutti i suoi metodi, sia vecchi che nuovi. Completata la modifica del tipo di dati, occorre modificarne il corpo, elenca-  
do tutti i metodi del tipo di dati in un unico comando.

Non è necessario concedere il privilegio EXECUTE sulle funzioni membro e sulle procedure dei tipi di dati astratti. Se si concede a un altro utente il privilegio EXECUTE sul tipo di dati ANIMALE\_TY, questo utente disporrà automaticamente del privilegio EXECUTE sui metodi che fanno parte del tipo di dati. Quindi la concessione dell'accesso al tipo di dati comprende sia la rappresentazione dei dati, sia i relativi metodi di accesso ai dati, in conformità alla filosofia dell'orientamento agli oggetti.

Quando si creano funzioni membro, è possibile definirle come metodi mappa o metodi ordi-  
ne (o non utilizzare nessuna delle due definizioni, come per la funzione membro ANIMA-

LE\_TY.Eta). Un *metodo mappa* restituisce la posizione relativa di un determinato record nell’ordinamento di tutti i record all’interno dell’oggetto. Il corpo di un tipo può contenere solo un metodo mappa, che deve essere una funzione.

Un *metodo ordine* è una funzione membro che accetta un record dell’oggetto come argomento esplicito e restituisce un valore intero. Se il valore restituito è negativo, zero o positivo, l’argomento implicito “autonomo” della funzione è rispettivamente minore, uguale o maggiore del valore del record specificato esplicitamente. Quando all’interno di un oggetto vengono confrontate più righe in una clausola *order by*, il metodo *ordine* viene eseguito automaticamente per ordinare le righe restituite. La specifica di un tipo di dati può contenere solo un metodo *ordine*, che dev’essere una funzione con tipo di restituzione INTEGER.

La nozione relativa all’ordinamento all’interno delle righe di un tipo di dati astratto può avere più importanza, se si prende in considerazione l’implementazione di collettori, tipi di dati che contengono più valori per riga. Oracle mette a disposizione due tipi di collettori, che prendono il nome di *tabelle annidate* e *array variabili*. Nel prossimo capitolo, si imparerà a implementare questi tipi di collettori.



## Capitolo 31

# Collettori (tabelle annidate e array variabili)

- 31.1 **Array variabili**
- 31.2 **Tabelle annidate**
- 31.3 **Problemi nella gestione  
di tabelle annidate e array variabili**

I collettori utilizzano tipi di dati astratti. Prima di usare gli array variabili e le tabelle annidate è consigliabile formarsi una conoscenza approfondita della creazione e implementazione di questi tipi di dati (Capitolo 4 e Capitolo 30).

### 31.1 Array variabili

Un array variabile consente di memorizzare attributi ripetuti di un record in un'unica riga. Per esempio, si supponga di voler tenere traccia degli attrezzi presi in prestito dai propri vicini. In un database relazionale, ciò può essere ottenuto creando una tabella PRESTITO:

```
create table PRESTITO
(Nome          VARCHAR2(25),
 Attrezzo      VARCHAR2(25),
 constraint PRESTITO_PK primary key (Nome, Attrezzo));
```

Anche se il valore Nome del dipendente non cambia, viene ripetuto in ciascun record in quanto fa parte della chiave primaria. I collettori come gli array variabili consentono di ripetere *solo* i valori di colonna che cambiano, permettendo un potenziale risparmio di spazio su disco. I collettori possono essere utilizzati per rappresentare in modo accurato le relazioni tra tipi di dati negli oggetti del database. Nei paragrafi seguenti, si imparerà a creare e usare i collettori.

#### Creazione di un array variabile

Un array variabile può essere creato basandosi sia su un tipo di dati astratto, sia su uno dei tipi di dati standard di Oracle (come NUMBER). Per esempio, si potrebbe creare un nuovo tipo di dati, ATTREZZO\_TY, come mostrato nel listato seguente, che possiede una sola colonna; è opportuno limitare gli array variabili a una colonna. Per utilizzare più colonne in un array, si prenda in considerazione la possibilità di utilizzare le tabelle annidate (descritte più avanti nel capitolo).

```
create type ATTREZZO_TY as object
(NomeAttrezzo  VARCHAR2(25));
```

Per creare un array variabile (cui verrà assegnato il nome ATTREZZI\_VA) non è necessario creare prima il tipo base, in questo caso ATTREZZO\_TY. Dato che l'array variabile si basa su un'unica colonna, è possibile creare il tipo ATTREZZI\_VA in un unico passaggio. Per creare l'array variabile, si utilizza la clausola `as varray()` del comando `create type`:

```
create or replace type ATTREZZI_VA as varray(5) of VARCHAR2(25);
```

Quando si esegue questo comando `create type`, viene creato un tipo di nome ATTREZZI\_VA. La clausola `as varray(5)` comunica a Oracle la creazione di un array variabile contenente un massimo di cinque voci per record. Il nome dell'array variabile è al plurale, ATTREZZI anziché ATTREZZO, per ricordare che questo tipo verrà utilizzato per contenere più record. Il suffisso VA chiarisce che il tipo è proprio un array variabile. Ora questo tipo potrà servire da base per la definizione di una colonna, come mostrato nel listato seguente.

```
create table PRESTITO
(Nome          VARCHAR2(25),
 Attrezzi      ATTREZZI_VA,
 constraint PRESTITO_PK primary key (Nome));
```

## Descrizione dell'array variabile

La tabella PRESTITO contiene un record per ciascuna persona che prende in prestito degli attrezzi, anche se questa persona ha in prestito più attrezzi. Questi attrezzi vengono memorizzati nella colonna Attrezzi con l'array variabile ATTREZZI\_VA. La tabella PRESTITO può essere descritta nel modo seguente:

```
desc PRESTITO
```

Name	Null?	Type
NOME	NOT NULL	VARCHAR2(25)
ATTREZZI		ATTREZZI_VA

Si osservi che il comando `set describe depth`, descritto nel Capitolo 30, non influisce su questo output. È possibile eseguire una query su informazioni simili da `USER_TAB_COLUMNS`:

```
select Column_Name,
       Data_Type
  from USER_TAB_COLUMNS
 where Table_Name = 'PRESTITO';
```

COLUMN_NAME	DATA_TYPE
NOME	VARCHAR2
ATTREZZI	ATTREZZI_VA

`USER_TYPES` mostra ATTREZZI\_VA, una collection priva di attributi:

```
select TypeCode,
       Attributes
  from USER_TYPES
 where Type_Name = 'ATTREZZI_VA';
```

TYPECODE	ATTRIBUTES
COLLEZIONE	0

Per ulteriori informazioni su ATTREZZI\_VA si esegua una query sulla vista del dizionario di dati USER\_COLL\_TYPES, come mostrato nel listato seguente.

```
select Coll_Type,
       Elem_Type_Owner,
       Elem_Type_Name,
       Upper_Bound,
       Length
  from USER_COLL_TYPES
 where Type_Name = 'ATTREZZI_VA'
```

COLL_TYPE	ELEM_TYPE_OWNER
ELEM_TYPE_NAME	UPPER_BOUND LENGTH
VARYING ARRAY	
VARCHAR2	5 25

Se si basa un array variabile su un tipo di dati astratto già esistente, il proprietario del tipo di dati viene visualizzato nella colonna Elem\_Type\_Owner e il nome del tipo di dati nella colonna Elem\_Type\_Name. Dato che ATTREZZI\_VA utilizza VARCHAR2 invece di un tipo di dati astratto, il valore della colonna Elem\_Type\_Owner è NULL.

## Inserimento di record nell'array variabile

Quando si crea un tipo di dati, il database crea automaticamente un metodo denominato *constructor method* per il tipo di dati. Come descritto nel Capitolo 30, il constructor method dev'essere utilizzato per inserire i record in colonne che utilizzano un tipo di dati astratto. Dato che un array variabile è un tipo di dati astratto, per inserire record in tabelle che utilizzano array variabili è necessario ricorrere ai constructor method. Inoltre, dato che l'array variabile è esso stesso un tipo di dati astratto, per inserire un record in una tabella che utilizza un array variabile può rendersi necessario annidare chiamate a più constructor method.

Le colonne della tabella PRESTITO sono Nome e Attrezzi (un array variabile che utilizza il tipo di dati ATTREZZI\_VA). Il comando seguente inserisce un singolo record nella tabella PRESTITO. In questo esempio, il record presenta un solo valore per la colonna Nome e tre valori per la colonna Attrezzi.

```
insert into PRESTITO values
('JED HOPKINS',
 ATTREZZI_VA('MARTELLO', 'MAZZA', 'ASCIA'));
```

Questo comando insert specifica subito il valore per la colonna Nome. Dato che questa colonna non fa parte di alcun tipo di dati astratto, viene popolata nello stesso modo in cui si inserirebbero dei valori in una qualsiasi colonna relazionale:

```
insert into PRESTITO values
('JED HOPKINS',
```

La parte successiva del comando `insert` inserisce dei record nella colonna Attrezzi. Dato che questa utilizza l'array variabile `ATTREZZI_VA`, è necessario utilizzare il constructor method `ATTREZZI_VA`:

```
ATTREZZI_VA('MARTELLO', 'MAZZA', 'ASCIA');
```

Dal momento che si tratta di un array variabile, è possibile passare più valori al constructor method `ATTREZZI_VA`. È esattamente quello che accade nel listato precedente; per il nome di un vicino sono elencati tre attrezzi in un unico inserimento. Se l'array variabile si fosse basato sul tipo di dati `ATTREZZO_TY`, le chiamate al tipo di dati `ATTREZZO_TY` sarebbero state annidate all'interno delle chiamate a `ATTREZZI_VA`:

```
insert into PRESTITO values
('JED HOPKINS',
ATTREZZI_VA(
    ATTREZZO_TY('MARTELLO'),
    ATTREZZO_TY('MAZZA'),
    ATTREZZO_TY('ASCIA')));
```

Se si basa l'array variabile su un tipo di dati fornito da Oracle (come `NUMBER`) invece che su un tipo di dati astratto o definito dall'utente, si semplifica il comando `insert` eliminando le chiamate annidate a più constructor method.

L'array variabile `ATTREZZI_VA` è stato definito in modo da avere sino a cinque valori. In questo comando `insert` sono stati inseriti nella riga soltanto tre valori, gli altri due valori rimangono così non inizializzati. Se si desidera impostare i valori non inizializzati a `NULL`, è sufficiente specificarlo nel comando `insert`:

```
insert into PRESTITO values
('JED HOPKINS',
ATTREZZI_VA('MARTELLO', 'MAZZA', 'ASCIA', NULL, NULL));
```

Se si specificano troppi valori da inserire nell'array variabile, Oracle risponde con:

```
ORA-22909: exceeded maximum VARRAY limit
```

Non è possibile inserire dei record in una tabella che contiene un array variabile, a meno di non conoscere la struttura e i tipi di dati all'interno dell'array. Questo esempio assumeva che tutti i tipi di dati utilizzati dalla tabella `PRESTITO` fossero contenuti nello schema usato per creare la stessa tabella `PRESTITO`. Se i tipi di dati sono di proprietà di un altro schema, si dovrà concedere al creatore della tabella `PRESTITO` il privilegio `EXECUTE` su questi tipi di dati. Durante gli inserimenti, il creatore della tabella `PRESTITO` deve specificare i proprietari dei tipi di dati. Per esempi sull'uso dei tipi di dati creati in altri schemi, si consulti il Capitolo 30.

## **Selezione di dati dagli array variabili**

Quando si inseriscono dei record in un array variabile, è necessario assicurarsi che il numero di voci inserite non superi il numero massimo consentito per l'array variabile. Il numero massimo di voci viene specificato durante la creazione dell'array variabile, e su questo numero si può eseguire una query da `USER_COLL_TYPES`, come descritto nel paragrafo precedente del capitolo.

Inoltre, è possibile eseguire una query sull'array variabile per stabilirne il numero massimo di voci per riga, chiamato `LIMIT`, e il numero corrente di voci per riga, chiamato `COUNT`.

Tuttavia, questa query non può essere eseguita direttamente tramite un comando SQL select. Per recuperare i valori di COUNT, LIMIT e i dati da un array variabile, è necessario ricorrere a PL/SQL. Si può utilizzare un gruppo di cicli FOR a cursore annidati, come mostrato nel listato seguente.

**NOTA** *Per la descrizione del linguaggio PL/SQL e dei cicli FOR a cursore si rimanda al Capitolo 27.*

```
set serveroutput on
declare
    cursor prestito_cursor is
        select * from PRESTITO;
begin
    for prestito_rec in prestito_cursor
    loop
        dbms_output.put_line('Nome Contatto: '||prestito_rec.Nome);
        for i in 1..prestito_rec.Attrezzi.Count
        loop
            dbms_output.put_line(prestito_rec.Attrezzi(i));
        end loop;
    end loop;
end;
```

Ecco l'output di questo script PL/SQL:

```
Nome Contatto: JED HOPKINS
MARTELLO
MAZZA
ASCIA
```

PL/SQL procedure successfully completed.

Lo script implementa diverse caratteristiche di PL/SQL per recuperare i dati dall'array variabile. I dati vengono visualizzati tramite la procedura PUT\_LINE del package DBMS\_OUTPUT (Capitolo 29). Per prima cosa è necessario abilitare la visualizzazione dell'output PL/SQL.

```
set serveroutput on
```

Nella sezione delle dichiarazioni del blocco PL/SQL viene dichiarato un cursore. La query del cursore seleziona tutti i valori della tabella PRESTITO:

```
declare
    cursor prestito_cursor is
        select * from PRESTITO;
```

La sezione dei comandi eseguibili del blocco PL/SQL contiene due cicli FOR annidati. Nel primo ciclo viene valutato ciascun record del cursore mentre il valore di Nome viene visualizzato tramite la procedura PUT\_LINE:

```
begin
    for prestito_rec in prestito_cursor
    loop
        dbms_output.put_line('Nome Contatto: '||prestito_rec.Nome);
```

**NOTA** *Sarebbe opportuno visualizzare soltanto dati caratteri tramite PUT\_LINE. PL/SQL visualizza i valori numerici tramite la procedura PUT\_LINE, tuttavia non li si può concatenare alle espressioni prima di averli visualizzati. Per questo motivo, potrebbe rendersi necessario l'uso della funzione TO\_CHAR sui dati numerici prima di visualizzarli tramite PUT\_LINE.*

Nella sezione successiva del blocco PL/SQL, le righe vengono valutate ulteriormente, per recuperare i dati memorizzati nell'array variabile. Quindi viene eseguito un secondo ciclo FOR, annidato all'interno del primo ciclo. Il ciclo FOR interno è limitato dal numero di valori nell'array, come definito dal metodo COUNT (il numero massimo di valori nell'array è disponibile nel metodo LIMIT).

I valori nell'array ATTREZZI vengono visualizzati, uno per uno, dal ciclo seguente:

```
for i in 1..prestito_rec.Attrezzi.Count
loop
    dbms_output.put_line(prestito_rec.Attrezzi(i));
```

### La notazione

Attrezzi(i)

comunica a Oracle di visualizzare il valore dell'array corrispondente al valore di *i*. Così alla prima esecuzione del ciclo viene visualizzato il primo valore contenuto nell'array; poi il valore di *i* viene incrementato e viene visualizzato il secondo valore dell'array.

L'ultima parte del blocco PL/SQL chiude i due cicli annidati e conclude il blocco stesso:

```
end loop;
end loop;
end;
```

Dato che questo metodo richiede cicli annidati all'interno di blocchi PL/SQL, la selezione di dati dagli array variabili non è banale.

Per semplificare il processo di selezione dei dati dagli array variabile, si può ricorrere alla funzione TABLE nella clausola from. Si consideri la query seguente e il suo output:

```
select B.Nome, N.*
  from PRESTITO B, TABLE(B.Attrezzi) N;
```

NAME	COLUMN_NAME
JED HOPKINS	MARTELLO
JED HOPKINS	MAZZA
JED HOPKINS	ASCIA

Come funziona? La clausola TABLE accetta come input il nome dell'array variabile mentre all'output viene assegnato l'alias *N*. Vengono selezionati i valori in *N*, creando così la seconda colonna dell'output. Questo metodo è più semplice di quello PL/SQL, tuttavia a causa della sintassi unica potrebbe essere più fuorviante per gli utenti.

Si osservi la possibilità di ottenere un numero diverso di attrezzi per ciascuna persona, un numero compreso tra zero e cinque. Per esempio, la riga seguente inserirebbe nell'array variabile una riga con una sola voce:

```
insert into PRESTITO values
('FRED FULLER',
ATTREZZI_VA('MARTELLO'));
```

Si può selezionare questo dato tramite PL/SQL, la funzione TABLE o con SQL:

```
select * from PRESTITO;
NAME
-----
ATTREZZI
-----
JED HOPKINS
ATTREZZI_VA('MARTELLO', 'MAZZA', 'ASCIA')

FRED FULLER
ATTREZZI_VA('MARTELLO')
```

Per una maggiore flessibilità nella selezione di dati dai collettori, è consigliato l'uso di tabelle annidate, come descritto nel paragrafo successivo.

## 31.2 Tabelle annidate

Mentre gli array variabili hanno un numero limitato di voci, un secondo tipo di collettore, le *tabelle annidate*, non prevede limiti al numero di voci per riga. Una tabella annidata è, come suggerisce il nome stesso, una tabella contenuta all'interno di un'altra. In questo caso, si tratta di una tabella rappresentata sotto forma di colonna in un'altra tabella. È possibile avere diverse righe della tabella interna associate a ciascuna riga della tabella principale.

Per esempio, se si ha una tabella di allevatori di animali, è possibile disporre anche di dati riguardanti gli animali dell'allevamento. Con una tabella annidata, si potrebbero memorizzare le informazioni sugli allevatori e sui loro animali all'interno della stessa tabella. Si consideri il tipo di dati ANIMALE\_TY presentato nel Capitolo 30:

```
create or replace type ANIMALE_TY as object
(Generazione      VARCHAR2(25),
Nome             VARCHAR2(25),
DataNascita      DATE);
```

**NOTA** Per semplicità, questa versione del tipo di dati ANIMALE\_TY non comprende alcun metodo.

**NOTA** Per questo esempio, si dovrebbe eliminare la tabella ANIMALE nel caso in cui esista già nel proprio schema Pratica.

Il tipo di dati ANIMALE\_TY contiene un record per ciascun animale: la sua genealogia, il nome e la data di nascita. Per utilizzare questo tipo di dati come base di una tabella annidata, è necessario creare un nuovo tipo di dati:

```
create type ANIMALI_NT as table of ANIMALE_TY;
```

La clausola `as table of` di questo comando `create type` indica a Oracle che questo tipo di dati servirà come base per una tabella annidata. Il nome del tipo di dati, ANIMALI\_NT, ha una

radice al plurale per indicare che memorizza più righe, e presenta il suffisso ‘NT’ per indicare che si tratta di una tabella annidata.

Ora è possibile creare una tabella di allevatori con il tipo di dati ANIMALI\_NT:

```
create table ALLEVATORE
(NomeAllevatore    VARCHAR2(25),
 Animali           ANIMALI_NT)
nested table ANIMALI store as ANIMALI_NT_TAB;
```

In questo comando `create table` viene creata la tabella `ALLEVATORE`. La prima colonna di questa tabella è `NomeAllevatore`. La seconda colonna è `Animali`, la cui definizione è la tabella annidata `ANIMALI_NT`:

```
create table ALLEVATORE
(NomeAllevatore    VARCHAR2(25),
 Animali           ANIMALI_NT)
```

Quando si crea una tabella che include una tabella annidata, è necessario specificare il nome della tabella utilizzata per memorizzare i dati della tabella annidata. In altre parole, i dati della tabella annidata non sono memorizzati “in linea” con i restanti dati della tabella. Al contrario, vengono memorizzati separatamente rispetto alla tabella principale. I dati nella colonna `Animali` vengono memorizzati in una tabella, mentre i dati nella colonna `NomeAllevatore` sono contenuti in una tabella separata. Oracle gestirà i puntatori tra le tabelle. In questo esempio, i dati “fuori linea” della tabella annidata vengono memorizzati in una tabella di nome `ANIMALI_NT_TAB`:

```
nested table ANIMALI store as ANIMALI_NT_TAB;
```

Questo esempio mette in evidenza l’importanza degli standard di denominazione per i collettori. Se i nomi delle colonne vengono espressi al plurale (“`Animali`” anziché “`Animale`”), e si utilizzano coerentemente i suffissi (“`TY`”, “`NT`” e così via), è possibile stabilire la natura di un oggetto semplicemente in base al suo nome. Inoltre, dato che le tabelle annidate e gli array variabili possono basarsi direttamente su tipi di dati definiti in precedenza, i nomi dei primi possono rispecchiare i nomi di questi ultimi. Così, un tipo di dati di nome `ANIMALE_TY` potrebbe essere utilizzato come base per un array variabile `ANIMALI_VA` o una tabella annidata `ANIMALI_NT`. La tabella annidata avrebbe una tabella fuori linea a essa associata, di nome `ANIMALI_NT_TAB`. Se si è a conoscenza dell’esistenza di una tabella `ANIMALI_NT_TAB` e si mantiene una certa coerenza nella denominazione dei propri oggetti, si saprà automaticamente che questa tabella memorizza dati basati su un tipo di dati `ANIMALE_TY`.

## Inserimento di record in una tabella annidata

È possibile inserire dei record in una tabella annidata grazie ai constructor method del tipo di dati della tabella. Per la colonna `Animali`, il tipo di dati è `ANIMALI_NT`; quindi, si dovrà ricorrere al constructor method `ANIMALI_NT`. Il tipo di dati `ANIMALI_NT` utilizza, a sua volta, il tipo di dati `ANIMALE_TY`. Come descritto nell’esempio seguente, l’inserimento di un record nella tabella `ALLEVATORE` richiede l’uso di entrambi i constructor method, `ANIMALI_NT` e `ANIMALE_TY`. Nell’esempio, per l’allevatore Jane James sono elencati tre animali.

```
insert into ALLEVATORE values
('JANE JAMES',
 ANIMALI_NT(

```

```

ANIMALE_TY('CANE', 'BUTCH', '31-MAR-01'),
ANIMALE_TY('CANE', 'ROVER', '05-JUN-01'),
ANIMALE_TY('CANE', 'JULIO', '10-JUN-01')
));

```

Questo comando `insert` specifica per prima cosa il nome dell'allevatore.

```
insert into ALLEVATORE values
('JANE JAMES',
```

Successivamente, viene inserito il valore della colonna `Animali`. Dato che questa colonna utilizza la tabella annidata `ANIMALI_NT`, viene chiamato il constructor method `ANIMALI_NT`:

```
ANIMALI_NT(
```

La tabella annidata `ANIMALI_NT` utilizza il tipo di dati `ANIMALE_TY`, quindi per ogni valore inserito viene chiamato il constructor method `ANIMALE_TY`:

```

ANIMALE_TY('CANE', 'BUTCH', '31-MAR-01'),
ANIMALE_TY('CANE', 'ROVER', '05-JUN-01'),
ANIMALE_TY('CANE', 'JULIO', '10-JUN-01')
));

```

Con il comando `describe` si possono visualizzare i tipi di dati che costituiscono la tabella e la sua tabella annidata. Per abilitare la visualizzazione degli attributi dei tipi di dati, si utilizza il comando SQL\*Plus `set describe depth`.

```
set describe depth 2
desc allevatore
```

Name	Null?	Type
NOMEALLEVATORE		VARCHAR2(25)
ANIMALI		ANIMALI_NT
GENERAZIONE		VARCHAR2(25)
 NOME		VARCHAR2(25)
 DATANASCITA		DATE

È possibile eseguire una query su questi metadati dalle viste del dizionario dati `USER_TAB_COLUMNS`, `USER_TYPES`, `USER_COLL_TYPES` e `USER_TYPE_ATTRS`, come mostrato negli esempi precedenti di questo capitolo.

## Query su tabelle annidate

Le tabelle annidate supportano una vasta gamma di query. Una tabella annidata è una colonna all'interno di una tabella. Per supportare le query delle colonne e righe di una tabella annidata, si utilizza la funzione `TABLE` introdotta in precedenza nel capitolo. La clausola `THE` adoperata nelle precedenti versioni di Oracle è ancora supportata per motivi di compatibilità, ma il suo impiego è ormai disapprovato.

Per comprendere come viene utilizzata la funzione TABLE, per prima cosa si consideri la tabella annidata. Se si trattasse di una colonna normale di una tabella relazionale, si potrebbe eseguire una query tramite un normale comando select:

```
select DataNascita /* Così non funziona.*/
  from ANIMALI_NT
 where Nome = 'JULIO';
```

ANIMALI\_NT non è una tabella normale, bensì un tipo di dati. Per selezionare delle colonne (come DataNascita) dalla tabella annidata, è innanzi tutto necessario “rimuovere” la tabella, per poterla sottoporre a query. Per prima cosa, si applichi la funzione TABLE alla colonna della tabella annidata (ALLEVATORE.Animali), come mostrato nel listato seguente:

```
select NomeAllevatore, N.Nome, N.DataNascita
  from ALLEVATORE, TABLE(ALLEVATORE.Animali) N;
```

NOMEALLEVATORE	NOME	DATANASCITA
JANE JAMES	BUTCH	31-MAR-01
JANE JAMES	ROVER	05-JUN-01
JANE JAMES	JULIO	10-JUN-01

Con i dati rappresentati in questo formato, è facile recuperare la riga corretta:

```
select NomeAllevatore, N.Nome, N.DataNascita
  from ALLEVATORE, TABLE(ALLEVATORE.Animali) N
 where N.Nome = 'JULIO';
```

NOMEALLEVATORE	NOME	DATANASCITA
JANE JAMES	JULIO	10-JUN-01

### Inserimenti, aggiornamenti e cancellazioni con la funzione TABLE

Ogni volta che si devono eseguire inserimenti o aggiornamenti direttamente nella tabella annidata, si può utilizzare la funzione TABLE. Per esempio, con il passare del tempo è probabile che un allevatore abbia sempre più animali, quindi si dovrà essere in grado di aggiungere nuovi record alla tabella annidata ANIMALI\_NT all'interno della tabella ALLEVATORE. Si dovranno inserire i record nella tabella annidata senza aggiungerne di nuovi alla tabella ALLEVATORE e correlare i nuovi record Animali agli allevatori già presenti nella tabella ALLEVATORE.

La seguente istruzione insert aggiunge un nuovo record alla tabella annidata nel record ALLEVATORE di Jane James:

```
insert into
  TABLE(select Animali
        from ALLEVATORE
       where NomeAllevatore = 'JANE JAMES')
values
  (ANIMALE_TY('CANE', 'MARCUS', '01-AUG-01'));
```

L'inserimento non fornisce un valore per la colonna NomeAllevatore, in quanto viene semplicemente eseguito un inserimento della riga di una allevatore nella tabella annidata. La funzione TABLE viene utilizzata nuovamente per “rimuovere” la tabella annidata nella colonna Animali (per la riga di Jane) e i nuovi valori vengono inseriti. Si osservi l'uso della clausola where all'interno della chiamata alla funzione TABLE per specificare le righe su cui si deve agire.

Si può anche aggiornare la data di nascita del cane. In origine, il compleanno di Julio era stato inserito come 10-JUN-01 ma alla fine si è scoperto che cadeva il 01-SEP-01. Il seguente aggiornamento modifica il valore DataNascita per Julio nella tabella annidata. La sua sintassi e struttura sono molto simili agli esempi delle query mostrati in precedenza.

```
update TABLE(select Animali
              from ALLEVATORE
              where NomeAllevatore = 'JANE JAMES') N
    set N.DataNascita = '01-SEP-01'
   where N.Nome = 'JULIO';
```

Si può verificare questa modifica eseguendo una query sui dati della tabella annidata:

```
select NomeAllevatore, N.Nome, N.DataNascita
      from ALLEVATORE, TABLE(ALLEVATORE.Animali) N;
```

NOMEALLEVATORE	NOME	DATANASCITA
JANE JAMES	BUTCH	31-MAR-01
JANE JAMES	ROVER	05-JUN-01
JANE JAMES	JULIO	01-SEP-01
JANE JAMES	MARCUS	01-AUG-01

Le voci della tabella annidata possono essere anche eliminate:

```
delete TABLE(select Animali
              from ALLEVATORE
              where NomeAllevatore = 'JANE JAMES') N
   where N.Nome = 'JULIO';
```

1 row deleted.

```
select NomeAllevatore, N.Nome, N.DataNascita
      from ALLEVATORE, TABLE(ALLEVATORE.Animali) N;
```

NOMEALLEVATORE	NOME	DATANASCITA
JANE JAMES	BUTCH	31-MAR-01
JANE JAMES	ROVER	05-JUN-01
JANE JAMES	MARCUS	01-AUG-01

### 31.3 Problemi nella gestione di tabelle annidate e array variabili

Le attività amministrative necessarie per le tabelle annidate e gli array variabili presentano alcune differenze, descritte nei paragrafi seguenti, insieme ad alcuni consigli grazie ai quali il tipo di collettore può rispondere nel modo migliore ai criteri.

#### Gestione di collection di grandi dimensioni

Quando si memorizzano dei dati correlati a una tabella, si può scegliere tra uno di questi metodi: un array variabile, una tabella annidata oppure una tabella separata. Se il numero delle righe è

limitato, può essere più appropriato un array variabile. Man mano che il numero delle righe aumenta, è possibile imbattersi in problemi di prestazioni mentre si accede all'array variabile. Questi problemi possono essere causati da una caratteristica degli array variabili: l'impossibilità di indicizzarli. Per contro, le tabelle annidate e le tabelle relazionali standard possono essere indicizzate. Per questi motivi, le prestazioni di un collettore array possono peggiorare con l'aumento delle sue dimensioni.

**NOTA** *A partire da Oracle9i, le tablespace trasportabili potranno disporre di array variabili.*

Le prestazioni degli array variabili sono penalizzate ulteriormente dalle difficoltà nell'esecuzione di query sui dati. Se gli array variabili non rappresentano una buona soluzione per l'applicazione, quale usare tra le altre due alternative: una tabella annidata o una tabella relazionale standard? Dipende. Le tabelle annidate e le tabelle relazionali servono a scopi differenti, quindi la scelta dipende da ciò che si desidera fare dei dati. Di seguito sono descritte le differenze principali:

- Le tabelle annidate sono tipi di dati astratti, creati mediante il comando `create type`. Per questo motivo, a esse possono essere associati dei metodi. Se di prevede di allegare metodi ai dati, allora sarebbe meglio utilizzare tabelle annidate al posto di tabelle relazionali. In alternativa, si potrebbe prendere in considerazione l'eventualità di adoperare viste oggetto con metodi, come descritto nel Capitolo 30.
- Le tabelle relazionali possono essere facilmente correlate ad altre tabelle relazionali. Se i dati possono essere correlati a molte altre tabelle, sarebbe meglio evitare di annidarli all'interno di una tabella. La memorizzazione dei dati in una propria tabella mette a disposizione la maggiore flessibilità possibile nella gestione delle relazioni tra i dati.

## Variabilità dei collettori

Come si è spiegato nel Capitolo 4, i tipi di dati astratti possono agevolare l'imposizione di standard all'interno del database. Tuttavia, quando si usano i collettori non è sempre possibile ottenere gli stessi vantaggi nella standardizzazione. Si consideri l'array variabile `ATTREZZI_VA` descritto in precedenza. Se un'altra applicazione o tabella desidera utilizzare un array variabile `ATTREZZI_VA`, dovrà adottare la stessa definizione per l'array, compreso lo stesso numero massimo di valori. La nuova applicazione può avere però un valore differente per il numero massimo di attrezzi. Pertanto l'array deve essere modificato in modo da supportare il maggior numero possibile di valori che è in grado di contenere in una qualsiasi delle sue implementazioni. Da ciò può risultare una diminuzione nell'utilità dei poteri di limitazione dell'array stesso. Per supportare più limiti di array, si dovrebbero creare più array, rinunciando così ai vantaggi, derivanti dal riutilizzo degli oggetti, di cui si potrebbe godere con la creazione di array variabili. Quindi, si dovranno gestire i differenti limiti degli array e conoscere il limite di ogni singolo array.

Si potrebbe anche scoprire che più tabelle annidate sono simili, con solo una o due colonne differenti. Per questo motivo, si può essere tentati di creare un unico tipo di tabella annidata che comprenda tutto quanto venga utilizzato in più tabelle, ma anche una tabella simile creerebbe problemi di gestione. Risulta infatti necessario sapere quali colonne della tabella annidata sono valide per le differenti implementazioni. Se non si può utilizzare la stessa struttura di colonna, con lo stesso significato e la stessa rilevanza delle colonne in ciascuna istanza, non si dovrebbe utilizzare come tipo di dati una tabella annidata condivisa. Se si desidera utilizzare comunque le

tabelle annidate in questo ambiente, è necessario creare un tipo di dati di tabella annidata separato per ciascuna istanza, in cui si utilizza la tabella annidata, e gestire questi tipi di dati separatamente. La presenza di più tipi di dati di tabella annidata complica la gestione del database.

## Collocazione dei dati

In un array variabile, i dati dell'array sono memorizzati con i rimanenti dati della tabella. Nelle tabelle annidate invece i dati possono essere memorizzati fuori linea. Di conseguenza, la quantità di dati analizzati durante le query che non riguardano i dati del collettore può essere inferiore per una tabella annidata rispetto a un array variabile. In altre parole, durante una query il database non avrà bisogno di leggere tutti i dati nella tabella annidata; se si utilizza un array variabile, il database potrà limitarsi a leggere i dati dell'array variabile per trovare ciò che cerca. I dati fuori linea, così come impiegati dalle tabelle annidate, possono migliorare le prestazioni delle query.

I dati fuori linea dei collezionatori emulano il modo in cui i dati verrebbero memorizzati in una normale applicazione di database relazionale. Per esempio, in un'applicazione relazionale i dati correlati (come dipendenti e capacità) verrebbero memorizzati in tabelle separate, e queste verrebbero memorizzate fisicamente in locazioni separate. Le tabelle annidate fuori linea memorizzano i propri dati separatamente dalla tabella principale, pur conservando con essa una relazione. I dati fuori linea possono migliorare le prestazioni delle query distribuendo l'I/O eseguito su una tabella tra più tabelle separate.

I dati fuori linea vengono utilizzati anche per memorizzare gli oggetti di grandi dimensioni (LOB) memorizzati internamente (a differenza dei BFILE che sono puntatori a LOB esterni al database). Nel prossimo capitolo si imparerà a creare e manipolare valori LOB sino a 4 GB di dimensione. La combinazione di tipi di dati astratti, viste oggetto, collezionatori e LOB mette a disposizione fondamenta solide per l'implementazione di un'applicazione di database relazionale a oggetti. Nel Capitolo 33 si imparerà ad ampliare ulteriormente questi oggetti per andare verso la creazione di un database a oggetti.



## Capitolo 32

# Uso di dati LOB

- 32.1 **Tipi di dati disponibili**
- 32.2 **Definizione dei parametri di memorizzazione per i dati LOB**
- 32.3 **Gestione e selezione dei valori LOB**

Per memorizzare dati carattere fino a 2 GB di lunghezza per riga è possibile utilizzare un tipo di dati LONG. Il tipo di dati LONG RAW consente invece la memorizzazione di dati binari lunghi. Al posto di LONG e LONG RAW, si possono usare i tipi di dati LOB (BLOB, CLOB, NCLOB, e BFILE) per memorizzare dati che raggiungono la lunghezza considerevole di 4GB. Anche se continua a consentire la creazione di colonne LONG e LONG RAW, Oracle consiglia di creare nuove colonne con i tipi di dati LOB e di convertire le colonne LONG e LONG RAW esistenti rispettivamente in colonne CLOB o BLOB. Se si utilizza uno di questi tipi di dati per memorizzare oggetti di grandi dimensioni (LOB), si potranno sfruttare anche le nuove possibilità di visualizzazione e manipolazione dei dati. Per eseguire ricerche di testo sui dati CLOB, si può usare anche Oracle Text (consultare il Capitolo 24).

In questo capitolo, viene descritto l'uso dei tipi di dati LOB e la manipolazione di dati, che non dovranno nemmeno essere memorizzati all'interno del database.

## 32.1 Tipi di dati disponibili

I tipi di dati LOB supportati sono quattro:

TIPO DI DATI LOB	DESCRIZIONE
BLOB	LOB binario; dati binari, fino a 4 GB di lunghezza memorizzati nel database.
CLOB	LOB carattere; dati carattere, fino a 4 GB di lunghezza memorizzati nel database.
BFILE	File binario; dati binari di sola lettura memorizzati al di fuori del database, la cui lunghezza è limitata dal sistema operativo.
NCLOB	Una colonna CLOB che supporta un set di caratteri a più byte.

Per ogni tabella si possono creare più LOB. Per esempio, si supponga di voler creare una tabella PROPOSTA per tenere traccia delle proposte formali inoltrate. I record delle proposte possono essere costituiti da una serie di file creati con elaboratori di testo e fogli di calcolo utilizzati per documentare e prezzare il lavoro proposto. La tabella PROPOSTA contiene tipi di dati VARCHAR2 (per colonne come quella relativa al nome del destinatario della proposta) e

tipi di dati LOB (contenenti i file creati con l'elaboratore di testo e il foglio di calcolo). Nel listato seguente, il comando `create table` crea la tabella PROPOSTA:

```
create table PROPOSTA
(Proposta_ID      NUMBER(10),
 Nome_Destin     VARCHAR2(25),
 Nome_Proposta   VARCHAR2(25),
 Breve_Descrizione VARCHAR2(1000),
 Testo_Proposta  CLOB,
 Budget          BLOB,
 Lettera_Present BFILE)
constraint PROPOSTA_PK primary key (Proposta_ID);
```

Dato che si possono sottoporre più proposte allo stesso destinatario, è necessario assegnare a ciascuna proposta un numero `Proposta_ID`. La tabella PROPOSTA memorizza il destinatario, il nome e una breve descrizione della proposta. Quindi, per sfruttare i tipi di dati LOB disponibili in Oracle si aggiungano tre colonne:

Testo_Proposta	CLOB contenente il testo della proposta.
Budget	BLOB contenente un foglio di calcolo che descrive i calcoli di costi e ricavi del lavoro proposto.
Lettera_Present	File binario (BFILE) memorizzato all'esterno del database, che contiene la lettera di presentazione allegata alla proposta.

In ogni riga della tabella PROPOSTA verranno memorizzati fino a tre valori LOB. Oracle sfrutterà le normali capacità del database per supportare l'integrità dei dati e della corrispondenza tra le voci `Proposta` e `Budget`, ma non per supportare i valori `Lettera_Present`. Dato che la colonna `Lettera_Present` utilizza il tipo di dati `BFILE`, i suoi dati vengono memorizzati all'esterno del database. Al suo interno, il database memorizza solo un valore di locator che gli permette di individuare il file esterno. Oracle non garantisce l'integrità dei dati per i file `BFILE` memorizzati all'esterno del database. Inoltre, Oracle non conferma l'esistenza del file quando si inserisce un record con un tipo di dati `BFILE`. La corrispondenza e l'integrità dei dati sono gestite solamente per i LOB memorizzati internamente.

I dati per le colonne LOB, siano essi memorizzati all'interno o all'esterno del database, potrebbero non essere memorizzati fisicamente all'interno della tabella PROPOSTA. All'interno della tabella PROPOSTA, Oracle memorizza valori di locator che puntano alle posizioni dei dati. Per i tipi di dati `BFILE`, il puntatore del locator punta a un file esterno; per i tipi di dati `BLOB` e `CLOB`, esso punta a una locazione separata per i dati, creata dal database per conservare i dati LOB. Pertanto, i dati LOB non vengono necessariamente memorizzati direttamente insieme ai restanti dati della tabella PROPOSTA.

Memorizzazioni *fuori linea* dei dati come queste consentono al database di evitare l'analisi dei dati LOB ogniqualvolta venga effettuata la lettura di più righe del database. I dati LOB vengono letti solo quando è necessario: negli altri casi vengono letti solo i valori dei relativi puntatori di locator. Si possono specificare parametri di memorizzazione (tablespace e dimensionamento) per l'area utilizzata per contenere i dati LOB, come descritto nel prossimo paragrafo.

## 32.2 Definizione dei parametri di memorizzazione per i dati LOB

Quando si crea una tabella contenente una colonna LOB, è possibile specificare parametri di memorizzazione per l'area adibita ai dati LOB. Quindi, per una tabella che non ha colonne LOB, è possibile inserire una clausola `storage` nel comando `create table` (Capitolo 20). Se la tabella contiene almeno una colonna LOB, all'interno del comando `create table` diventa necessaria anche un'altra clausola `lob`.

Si consideri la tabella PROPOSTA, questa volta con clausole `storage` e `tablespace`:

```
create table PROPOSTA
(Proposta_ID      NUMBER(10) primary key,
Nome_Destin      VARCHAR2(25),
Nome_Proposta    VARCHAR2(25),
Breve_Descrizione VARCHAR2(1000),
Testo_Proposta   CLOB,
Budget           BLOB,
Lettera_Present  BFILE
constraint PROPOSTA_PK primary key (Proposta_ID))
storage (initial 50K next 50K pctincrease 0)
tablespace PROPOSTE;
```

Dal momento che sono presenti tre colonne LOB, il comando `create table` dev'essere modificato in modo da comprendere le specifiche di memorizzazione per i dati LOB fuori linea. Due delle colonne LOB, `Testo_Proposta` e `Budget`, memorizzano i dati all'interno del database. Nel listato seguente è riportata la versione aggiornata del comando `create table`, che specifica la clausola di memorizzazione dei LOB.

```
create table PROPOSTA
(Proposta_ID      NUMBER(10),
Nome_Destin      VARCHAR2(25),
Nome_Proposta    VARCHAR2(25),
Breve_Descrizione VARCHAR2(1000),
Testo_Proposta   CLOB,
Budget           BLOB,
Lettera_Present  BFILE)
constraint PROPOSTA_PK primary key (Proposta_ID))
storage (initial 50K next 50K pctincrease 0)
tablespace PROPOSTE
lob (Testo_Proposta, Budget) store as
(tablespace Proposta_Lobs
storage (initial 100K next 100K pctincrease 0)
chunk 16K pctversion 10 nocache logging);
```

Le ultime quattro linee del comando `create table` specificano a Oracle in che modo memorizzare i dati LOB fuori linea. Le istruzioni sono le seguenti:

```
lob (Testo_Proposta, Budget) store as
(tablespace Proposta_Lobs
storage (initial 100K next 100K pctincrease 0)
chunk 16K pctversion 10 nocache logging);
```

La clausola lob specifica a Oracle che il gruppo di comandi successivo si riferisce alle specifiche di memorizzazione fuori linea per le colonne LOB della tabella. Le due colonne LOB (Testo\_Proposta e Budget) sono elencate esplicitamente. Quando i valori delle due colonne sono memorizzati fuori linea vengono registrati in un segmento, come specificato dalla clausola lob:

```
lob (Testo_Proposta, Budget) store as
```

**NOTA** *Se la colonna LOB fosse una sola, si potrebbe specificare un nome per il segmento LOB. Il nome del segmento verrebbe riportato subito dopo la clausola store as nel comando specificato precedentemente.*

Le due linee successive specificano i parametri della tablespace e della memorizzazione fuori linea per i LOB. La clausola tablespace assegna i dati fuori linea alla tablespace PROPOSTA\_LOBS, mentre la clausola storage serve per specificare i valori initial, next e pctincrease utilizzati per la memorizzazione fuori linea dei dati. Per una spiegazione dei parametri di memorizzazione disponibili per i vari segmenti, si rimanda alla voce della clausola storage nel Capitolo 42.

```
(tablespace Proposta_Lobs
storage (initial 100K next 100K pctincrease 0)
```

Dato che i segmenti sono adibiti alla memorizzazione di dati LOB, sono disponibili molti altri parametri, specificati all'interno della clausola lob:

```
chunk 16K pctversion 10
```

Il parametro di memorizzazione di LOB chunk indica a Oracle quanto spazio allocare durante ogni manipolazione di valori LOB. La dimensione predefinita per chunk è di 1K e può raggiungere un massimo di 32K.

Il parametro pctversion rappresenta la percentuale massima di spazio di memorizzazione complessivo di LOB utilizzato per la creazione di nuove versioni del LOB. Per default, le versioni precedenti dei dati LOB non vengono sovrascritte fino a che non viene utilizzato il 10 per cento dello spazio di memorizzazione di LOB disponibile.

Gli ultimi due parametri relativi allo spazio di memorizzazione di LOB indicano a Oracle come gestire i dati LOB durante le letture e le scritture:

```
nocache logging);
```

Il parametro nocache nella clausola di memorizzazione lob specifica che i valori LOB non vengono conservati in memoria per consentire un accesso più rapido durante le query. L'impostazione di memorizzazione predefinita di lob è nocache; il suo opposto è cache.

Il parametro logging specifica che tutte le operazioni effettuate sui dati LOB verranno registrate nei redo log file del database. Al contrario di logging, nologging impedisce che le operazioni vengano registrate nei redo log file, migliorando le prestazioni durante le operazioni di creazione delle tabelle. Inoltre, è possibile specificare i parametri tablespace e storage per un indice LOB. Per la sintassi completa degli indici LOB, si consulti la voce del comando create table nel Capitolo 42.

Ora che la tabella è stata creata, si può iniziare a inserirvi dei record. Nel prossimo paragrafo, viene descritto l'inserimento dei valori LOB nella tabella PROPOSTA e l'uso del package DBMS\_LOB per la manipolazione dei valori.

### 32.3 Gestione e selezione dei valori LOB

Esistono diversi modi per manipolare e selezionare i dati LOB. A partire da Oracle9i si potranno usare le funzioni per stringhe di caratteri sui tipi di dati LOB nello stesso modo in cui le si utilizzerebbe sulle colonne di tipo VARCHAR2. Per valori LOB grandi (100 KB di lunghezza), o per operazioni più complesse sui dati, sarebbe opportuno manipolare i dati LOB tramite il package DBMS\_LOB. Altri metodi per la manipolazione dei dati LOB comprendono l'uso di interfacce per la programmazione di applicazioni API (Application Programming Interface) e programmi OCI (Oracle Call Interface). In questo paragrafo viene descritto l'uso delle funzioni di stringa e si introduce il package DBMS\_LOB. Come dimostrano questi esempi, i LOB sono molto più flessibili dei tipi di dati LONG per quanto concerne le possibilità di manipolazione dei dati.

#### Inizializzazione dei valori

Si consideri nuovamente la tabella PROPOSTA. Proposta\_ID è la colonna di chiave primaria della tabella PROPOSTA. La tabella PROPOSTA contiene tre colonne LOB, una colonna BLOB, una CLOB e infine una BFILE. Per ciascuna delle colonne LOB, Oracle memorizza un *valore locator* che indica dove trovare qualsiasi dato fuori linea memorizzato per il record.

Quando si inserisce un record in una tabella che contiene LOB, ci si serve delle funzioni per comunicare a Oracle di creare un valore locator vuoto per le colonne LOB memorizzate internamente. Un valore locator vuoto è diverso da un valore NULL. Se il valore di una colonna LOB memorizzata internamente è NULL, è necessario impostarlo a un valore locator vuoto prima di aggiornarlo a un valore non-NULL.

Si supponga di iniziare a lavorare a una proposta e di inserire un record nella tabella PROPOSTA. A questo punto, non si dispone né di un foglio di calcolo per il budget, né di una lettera di presentazione. Il comando insert è visibile nel listato seguente:

```
insert into PROPOSTA
(Proposta_ID, Nome_Destin, Nome_Proposta, Breve_Descrizione,
Testo_Proposta,
Budget, Lettera_Present)
values
(1, 'DOT RODALE', 'MANTENIMENTO GIARDINO ORGANICO', NULL,
'Questo è il testo di una proposta per mantenere un giardino organico.'
EMPTY_BLOB(),NULL);
```

Il record inserito ha come Proposta\_ID un valore pari 1, e come Nome\_Destin 'DOT RODALE'. Il valore di Nome\_Proposta è MANTENIMENTO GIARDINO ORGANICO e la colonna Breve\_Descrizione per ora è lasciata a NULL. La colonna Testo\_Proposta è impostata per ora a una breve stringa di caratteri. Per impostare la colonna Budget a un valore locator vuoto, viene utilizzata la funzione EMPTY\_BLOB. Per impostare una colonna di tipo CLOB uguale a un valore locator vuoto, si ricorre alla funzione EMPTY\_CLOB. Dal momento che si tratta di un valore BFILE memorizzato esternamente, la colonna Lettera\_Present viene impostata a NULL.

Per impostare una colonna LOB a un valore locator vuoto, è necessario conoscere il tipo di dati della colonna:

BLOB	Utilizzare EMPTY_BLOB()
CLOB	Utilizzare EMPTY_CLOB()
NCLOB	Utilizzare EMPTY_CLOB()
BFILE	Utilizzare BFILENAME

Per puntare a una combinazione di directory e nome file, occorre utilizzare la procedura BFILENAME. Prima di specificare un valore per la directory nella funzione BFILENAME, un utente con il ruolo DBA o il privilegio di sistema CREATE ANY DIRECTORY dovrà creare la directory. Per creare una directory si utilizza il comando create directory, come nell'esempio seguente:

```
create directory dir_proposta as '/u01/proposal/letters';
```

Quando si inseriscono voci BFILE, si fa riferimento al nome della directory logica, come 'dir\_proposta', e non al nome della directory fisica nel sistema operativo. Gli utenti con autorità di DBA sono in grado di concedere agli utenti l'accesso READ ai nomi di directory. Per ulteriori informazioni, fare riferimento alla voce del comando create directory nel Capitolo 42.

Ora è possibile inserire un secondo record PROPOSTA con un valore per Lettera\_Present.

```
insert into PROPOSTA
(Proposta_ID, Nome_Destin, Nome_Proposta, Breve_Descrizione,
 Testo_Proposta, Budget,
 Lettera_Present)
values
(2, 'BRAD OHMONT', 'RICOSTRUIRE RECINZIONE', NULL,
 EMPTY_CLOB(), EMPTY_BLOB(),
 BFILENAME('dir_proposta','P2.DOC'));
```

In questo listato, nella tabella PROPOSTA viene inserita una seconda proposta: ricostruire una recinzione. Le funzioni EMPTY\_CLOB() ed EMPTY\_BLOB() vengono utilizzate per inserire valori locator vuoti nella tabella. La funzione BFILENAME indica a Oracle dove si trova esattamente la lettera di presentazione, ovvero nella directory 'dir\_proposta' e con il nome P2.DOC. All'interno di BFILENAME il primo parametro è sempre il nome della directory, mentre il secondo parametro è il nome del file contenuto nella directory.

**NOTA** *Non è necessario che il file P2.DOC esista per poter inserire correttamente questo record.*

Quando si seleziona il valore da una colonna LOB, Oracle usa il valore del locator per individuare i dati associati ai dati LOB. Non è mai necessario conoscere o specificare i valori del locator. Inoltre, come viene spiegato nei paragrafi successivi, i valori LOB consentono operazioni impossibili con i tipi di dati LONG.

## Inserimento con subquery

Come si procede se si desidera duplicare il record di una proposta? Per esempio, si supponga di iniziare a lavorare a una terza proposta, molto simile alla prima:

```
insert into PROPOSTA
(Proposta_ID, Nome_Destin, Nome_Proposta, Breve_Descrizione,
 Testo_Proposta, Budget, Lettera_Present)
select 3, 'SALTA GATES', 'AZZERAMENTO CAMPO GATES', NULL,
       Testo_Proposta, Budget, Lettera_Present
  from PROPOSTA
 where Proposta_ID = 1;
```

Questo comando `insert` ordina a Oracle di trovare il record `PROPOSTA` con valore `Proposta_ID` pari a 1. Si prendano i valori delle colonne `Testo_Proposta`, `Budget` e `Lettera_Present` e i valori letterali specificati (3, 'SALTA GATES' e così via) e si inserisca un nuovo record nella tabella `PROPOSTA`. Si osservi che le colonne inserite verranno impostate a valori di locator vuoti. La possibilità di eseguire inserimenti basandosi su query rappresenta un vantaggio significativo derivante dall'uso dei tipi di dati LOB. Questo tipo di inserimento non può essere utilizzato se una colonna `LONG` fa parte della query.

**NOTA** *Se si usa insert as select per inserire i valori LOB, è possibile che più valori BFILE puntino allo stesso file esterno. Potrebbe essere necessario aggiornare i nuovi valori in modo che puntino ai file esterni corretti. Oracle non gestisce l'integrità dei dati per i file esterni.*

## Aggiornamento dei valori LOB

Nel terzo record creato, il valore di `Testo_Proposta` è stato copiato dal primo record. Per aggiornare il valore `Testo_Proposta` del record che ha un valore di `Proposta_ID` pari a 3, si esegua il comando seguente:

```
update PROPOSTA
  set Testo_Proposta = 'Questo è il nuovo testo della proposta.'
 where Proposta_ID = 3;
```

Si può anche aggiornare la colonna `Lettera_Present` in modo che punti alla lettera di presentazione corretta. Per puntare al file corretto, si utilizza la funzione `BFILENAME`:

```
update PROPOSTA
  set Lettera_Present = BFILENAME('dir_proposta', 'P3.DOC')
 where Proposta_ID = 3;
```

È anche possibile aggiornare i valori `NULL` delle colonne `BFILE` senza doverle impostare prima a valori di locator vuoti. Se è stato inserito un record con un valore `NULL` per `Budget` (una colonna `BLOB`), non è possibile aggiornare il valore della colonna `Budget` se non si è prima provveduto a impostare il suo valore pari a quello restituito da `EMPTY_BLOB()`, mediante un comando `update`. Una volta impostato tale valore al valore restituito da `EMPTY_BLOB()` e stabilito un valore di locator vuoto, si potrà aggiornare il valore della colonna `Budget`.

## Uso delle funzioni di stringa per manipolare i valori LOB

A partire da Oracle9i, sui valori CLOB si potranno utilizzare le funzioni di stringa messe a disposizione da Oracle. Per esempio, per modificare le stringhe è possibile usare SUBSTR, INSTR, LTRIM e INITCAP. Per una descrizione delle funzioni di stringa incorporate, fare riferimento al Capitolo 7.

Nella tabella PROPOSTA, la colonna Testo\_Proposta è stata definita come tipo di dati CLOB. Ecco i tre valori di Testo\_Proposta (il secondo è NULL).

```
select Testo_Proposta from PROPOSTA;
```

```
TESTO_PROPOSTA
```

```
-----  
Questo è il testo di una proposta per mantenere un giardino organico.
```

```
Questo è il nuovo testo della proposta.
```

Se si applica INITCAP al terzo record si ottiene questo risultato:

```
select INITCAP(Testo_Proposta)  
      from PROPOSTA  
     where Proposta_ID = 3;
```

```
INITCAP(TESTO_PROPOSTA)
```

```
-----  
Questo È Il Nuovo Testo Della Proposta.
```

Se si applica INSTR a tutte tre le righe si ottiene il risultato previsto:

```
select INSTR(Testo_Proposta,'new',1,1) from PROPOSTA;
```

```
INSTR(TESTO_PROPOSTA,'NEW',1,1)
```

```
-----  
          0  
          0  
         13
```

La funzione SUBSTR può essere utilizzata per restituire i primi dieci caratteri del testo:

```
select SUBSTR(Testo_Proposta,1,10)  
      from PROPOSTA  
     where Proposta_ID = 3;
```

```
SUBSTR(TESTO_PROPOSTA,1,10)
```

```
-----  
Questo è i
```

Tuttavia alle colonne CLOB non è possibile applicare le funzioni NVL e DECODE:

```
select NVL(Testo_Proposta, "NULL") from PROPOSTA;  
*
```

```
ERROR at line 1:  
ORA-00932: inconsistent datatypes
```

Per le manipolazioni complesse, si dovrebbe adoperare il package DBMS\_LOB, come descritto nei prossimi paragrafi.

## **Uso di DBMS\_LOB per la manipolazione di valori LOB**

Il package DBMS\_LOB può essere utilizzato per modificare o selezionare valori LOB. Nei paragrafi seguenti, si imparerà a eseguire funzioni SQL come SUBSTR e INSTR sui valori LOB, nonché ad apportare aggiunte a valori LOB, a confrontare e a leggere questi valori.

Le procedure e le funzioni per il confronto e la manipolazione delle stringhe disponibili nel package DBMS\_LOB sono elencate nella Tabella 32.1.

Le procedure e le funzioni più importanti disponibili in questo package sono descritte nei paragrafi successivi. Quando si usano i BFILE, il numero massimo di file che è possibile tenere aperti contemporaneamente è limitato dall'impostazione del parametro SESSION\_MAX\_OPEN\_FILES nel file dei parametri di sistema del database. Il numero massimo predefinito di file BFILE contemporaneamente aperti è 10; il valore massimo può essere 50 o quello del parametro MAX\_OPEN\_FILES; in genere si usa il minore dei due.

Negli esempi seguenti, la colonna CLOB Testo\_Proposta viene utilizzata per mostrare l'effetto delle procedure e delle funzioni principali.

### **READ**

La procedura READ legge una parte di un valore LOB. Nell'esempio relativo alla tabella PROPOSTA, il testo di Testo\_Proposta per la prima riga è

```
select Testo_Proposta
  from PROPOSTA
 where Proposta_ID = 1;
```

TESTO\_PROPOSTA

-----  
Questo è il testo di una proposta per mantenere un giardino organico.

Questa descrizione è piuttosto breve e avrebbe potuto essere memorizzata in una colonna di tipo VARCHAR2. Dato che questa descrizione è memorizzata in una colonna CLOB, la sua lunghezza massima è comunque molto maggiore di quanto sarebbe stata se memorizzata in una colonna VARCHAR2; infatti può espandersi fino a 4 GB. Per i prossimi esempi, si utilizzerà questa breve stringa a scopi dimostrativi; le stesse funzioni e operazioni possono essere eseguite su stringhe CLOB più lunghe.

La procedura READ prevede quattro parametri, che devono essere specificati nell'ordine riportato di seguito:

1. Il locator del LOB (per un BLOB, CLOB o un BFILE).
2. Numero di byte o caratteri da leggere.
3. Offset (punto di inizio della lettura) dall'inizio del valore LOB.
4. I dati di output ricavati dalla procedura READ.

Se si raggiunge la fine del valore LOB prima che sia stato letto il numero di byte specificato, la procedura READ restituisce un errore.

La procedura READ richiede come input il locator LOB, e per questo motivo viene eseguita all'interno di blocchi PL/SQL. Si dovrà selezionare il valore del locator LOB, usarlo come input per la procedura READ quindi visualizzare il testo letto tramite il package DBMS\_OUTPUT.

**Tabella 32.1** Sottoprogrammi di DBMS\_LOB.

SOTTOPROGRAMMA	DESCRIZIONE
APPEND, procedura	Aggiunge il contenuto del LOB di origine al LOB di destinazione.
CLOSE, procedura	Chiude un LOB interno o esterno aperto in precedenza.
COMPARE, funzione	Confronta due LOB interi o parti di due LOB.
COPY, procedura	Copia tutto il LOB di origine, o parte di esso, nel LOB di destinazione.
CREATETEMPORARY, procedura	Crea un BLOB o CLOB temporaneo e il suo indice corrispondente nella tablespace temporanea predefinita dell'utente.
ERASE, procedura	Cancella tutto o parte di un LOB.
FILECLOSE, procedura	Chiude il file.
FILECLOSEALL, procedura	Chiude tutti i file aperti in precedenza.
FILEEXISTS, funzione	Verifica se il file esiste sul server.
FILEGETNAME, procedura	Ottiene l'alias della directory e il nome di file.
FILEISOPEN, funzione	Verifica se il file è stato aperto utilizzando i locator BFILE di input.
FILEOPEN, procedura	Apre un file.
FREETEMPORARY, procedura	Libera il BLOB o CLOB temporaneo nella tablespace temporanea predefinita dell'utente.
GETCHUNKSIZE, funzione	Restituisce la quantità di spazio utilizzata nel chunk del LOB per memorizzare il valore del LOB.
GETLENGTH, funzione	Ottiene la lunghezza del valore del LOB.
INSTR, funzione	Restituisce la posizione corrispondente della <i>ennesima</i> occorrenza del pattern nel LOB.
ISOPEN, funzione	Verifica se il LOB è già stato aperto utilizzando il locator di input.
ISTEMPORARY, funzione	Verifica se il locator punta a un LOB temporaneo.
LOADFROMFILE, procedura	Carica i dati BFILE in un LOB interno.
OPEN, procedura	Apre un LOB (interno, esterno o temporaneo) nella modalità indicata.
READ, procedura	Legge i dati dal LOB a partire dall'offset specificato.
SUBSTR, funzione	Restituisce parte del valore del LOB a partire dall'offset specificato.
TRIM, procedura	Taglia il valore del LOB alla lunghezza inferiore specificata.
WRITE, procedura	Scrive i dati nel LOB a partire dall'offset specificato.
WRITEAPPEND, procedura	Scrive un buffer alla fine di un LOB.

**Una nota sul package DBMS\_OUTPUT**

Come si è ricordato nel Capitolo 29, il package DBMS\_OUTPUT può essere utilizzato per visualizzare i valori di variabili all'interno di un blocco PL/SQL. La procedura PUT\_LINE al-

l'interno di DBMS\_OUTPUT visualizza il risultato specificato su una linea. Prima di utilizzare il package DBMS\_OUTPUT, sarebbe opportuno eseguire il comando set serveroutput on. Nel paragrafo successivo, si imparerà a creare il blocco PL/SQL che leggerà i dati da un LOB.

### Esempio di impiego della procedura READ

Per utilizzare la procedura READ è necessario conoscere il valore del locator del LOB che si desidera leggere. Il valore del locator dev'essere selezionato dalla tabella che contiene il LOB. Poiché il valore del locator dev'essere fornito come input alla procedura READ, è necessario utilizzare le variabili PL/SQL per contenerlo. La procedura READ, a sua volta, inserisce il proprio output in una variabile PL/SQL. Il package DBMS\_OUTPUT consente di visualizzare il valore di output. La struttura del blocco PL/SQL impiegato in questo esempio è la seguente:

```
declare
    variabile che contiene il valore del locator
    variabile che contiene la quantità (numero di caratteri/byte da leggere)
    variabile che contiene l'offset
    variabile che contiene l'output
begin
    set value for quantita_var;
    set value for offset_var;
    select locator value into locator var from table;
    DBMS_LOB.READ(locator var, quantita var, offset var, output var);
    DBMS_OUTPUT.PUT_LINE('Output:' || output var);
end;
```

**NOTA** Per ulteriori informazioni sui blocchi PL/SQL e i loro componenti, si rimanda al Capitolo 27.

Per la tabella PROPOSTA, la selezione dei primi dieci caratteri della colonna Testo\_Proposta assume l'aspetto seguente:

```
declare
    var_locator CLOB;
    quantita_var INTEGER;
    offset_var   INTEGER;
    var_output   VARCHAR2(10);
begin
    quantita_var := 10;
    offset_var := 1;
    select Testo_Proposta into var_locator
        from PROPOSTA
       where Proposta_ID = 1;
    DBMS_LOB.READ(var_locator, quantita_var, offset_var, var_output);
    DBMS_OUTPUT.PUT_LINE('Inizio del testo della proposta: ' || var_output);
end;
```

Quando si esegue il blocco PL/SQL precedente, l'output corrisponde a quello dell'esempio relativo alla funzione SQL SUBSTR, mostrato precedentemente:

Inizio del testo della proposta: Questo è i

PL/SQL procedure successfully completed.

L'output visualizza i primi dieci caratteri del valore Testo\_Proposta, a partire dal primo carattere del valore LOB.

Il blocco PL/SQL dell'esempio precedente dichiara in primo luogo le variabili da utilizzare:

```
declare
    var_locator    CLOB;
    quantita_var   INTEGER;
    offset_var     INTEGER;
    var_output     VARCHAR2(10);
```

Quindi, alle variabili vengono assegnati i relativi valori:

```
begin
    quantita_var := 10;
    offset_var := 1;
```

Il valore del locator è stato selezionato dalla tabella e memorizzato in una variabile di nome *var\_locator*:

```
select Testo_Proposta into var_locator
    from PROPOSTA
    where Proposta_ID = 1;
```

In seguito, viene chiamata la procedura READ usando i valori assegnati finora:

```
DBMS_LOB.READ(var_locator, quantita_var, offset_var, var_output);
```

Come risultato dell'esecuzione della procedura READ, la variabile *var\_output* viene popolata con i dati letti. La procedura PUT\_LINE visualizza questi dati:

```
DBMS_OUTPUT.PUT_LINE('Inizio del testo della proposta: ' || var_output);
```

Per la parte restante del capitolo verrà utilizzato questo formato dei blocchi PL/SQL. Facoltativamente, i valori delle variabili potranno essere impostati mentre queste vengono dichiarate.

Per selezionare una parte diversa dell'oggetto CLOB Proposta\_Testo, si modifichino le variabili di quantità e di offset. Se si modifica il numero di caratteri letti, è necessario modificare anche la dimensione della variabile di output. Nell'esempio seguente vengono letti 12 caratteri da Testo\_Proposta, partendo dal decimo carattere:

```
declare
    var_locator    CLOB;
    var_quantita   INTEGER;
    var_offset     INTEGER;
    var_output     VARCHAR2(12);
begin
    var_quantita := 12;
    var_offset := 10;
    select Testo_Proposta into var_locator
        from PROPOSTA
        where Proposta_ID = 1;
    DBMS_LOB.READ(var_locator, var_quantita, var_offset, var_output);
    DBMS_OUTPUT.PUT_LINE('Parte del testo della proposta: ' || var_output);
end;
```

Nel listato seguente, è riportato l'output prodotto da questo blocco PL/SQL.

Parte del testo della proposta: 1 testo di u

PL/SQL procedure successfully completed.

Se **Testo\_Proposta** fosse stata creata come una colonna VARCHAR2, sarebbe stato possibile selezionare questi dati con la funzione SUBSTR. Tuttavia, la lunghezza massima della colonna sarebbe stata limitata a 4.000 caratteri. Inoltre, si osservi che la colonna LOB è in grado di contenere dati binari (tipi di dati BLOB) e che la procedura READ può essere utilizzata per selezionare dati dagli oggetti BLOB nello stesso modo in cui si selezionano dati da oggetti CLOB. Per gli oggetti BLOB, è necessario dichiarare il buffer di output per utilizzare il tipo di dati RAW. In PL/SQL, i tipi di dati RAW e VARCHAR2 (utilizzati per le variabili di output delle letture CLOB e BLOB) hanno una lunghezza massima pari a 32767 caratteri.

## SUBSTR

La funzione SUBSTR all'interno del package DBMS\_LOB esegue la funzione SQL SUBSTR su un valore LOB. Questa funzione prevede tre parametri di input, che devono essere specificati in questo ordine:

1. Locator LOB.
2. Numero di byte o caratteri da leggere.
3. Offset (punto di inizio della lettura) dall'inizio del valore LOB.

Dato che SUBSTR è una funzione, non restituisce direttamente un valore per la variabile di output come avviene con READ. Per la procedura READ era stata dichiarata una variabile di output, quindi la si era popolata all'interno della chiamata alla procedura stessa. Il blocco PL/SQL utilizzato per la procedura READ è riportato nel listato seguente, con le linee relative alla variabile di output evidenziate in grassetto:

```
declare
    var_locator  CLOB;
    var_quantita   INTEGER;
    var_offset     INTEGER;
    var_output     VARCHAR2(12);
begin
    var_quantita := 12;
    var_offset := 10;
    select Testo_Proposta into var_locator
        from PROPOSTA
        where Proposta_ID = 1;
    DBMS_LOB.READ(var_locator, var_quantita, var_offset, var_output);
    DBMS_OUTPUT.PUT_LINE('Parte del testo della proposta: ' || var_output);
end;
```

Dal momento che SUBSTR è una funzione, la variabile di output dev'essere riempita in modo diverso. Il formato è

```
var_output := DBMS_LOB.SUBSTR(var_locator, var_quantita, var_offset);
```

L'istruzione PL/SQL precedente ricorre alla funzione SUBSTR del package DBMS\_LOB per selezionare 12 caratteri dalla colonna LOB **Testo\_Proposta**, a partire dal decimo. Nel listato seguente è mostrato l'output.

Sottostringa del testo della proposta: l testo di u  
 PL/SQL procedure successfully completed.

La chiamata alla funzione SQL equivalente è

```
select SUBSTR(Testo_Proposta,10,12)
  from PROPOSTA
 where Proposta_ID = 1;
```

Sono solo due le differenze tra l'esempio di SUBSTR e quello della procedura READ: il nome della chiamata alla funzione e la modalità di assegnamento di un valore alla variabile di output. Come nell'esempio della procedura READ, per la visualizzazione dei risultati viene utilizzata la procedura PUT\_LINE del package DBMS\_OUTPUT.

In generale, la funzione SUBSTR dovrebbe essere utilizzata quando si desidera selezionare solo una parte specifica del valore LOB. La funzione READ può servire per selezionare solo una parte di una stringa di testo (come negli esempi precedenti), tuttavia viene utilizzata più spesso all'interno di un ciclo. Per esempio, si supponga che il valore LOB sia maggiore della massima variabile RAW o VARCHAR2 consentita in PL/SQL (32.767 caratteri). In questo caso, si potrebbe utilizzare la procedura READ all'interno di un ciclo per leggere tutti i dati dal valore LOB. Per ulteriori informazioni sulle varie opzioni di ciclo disponibili in PL/SQL, si rimanda al Capitolo 27.

## **INSTR**

La funzione INSTR all'interno del package DBMS\_LOB esegue la funzione SQL INSTR su un valore LOB.

La funzione INSTR prevede quattro parametri di input che devono essere specificati in questo ordine:

1. Locator LOB.
2. Sequenza da verificare (byte RAW per i BLOB, stringhe di caratteri per i CLOB).
3. Offset (punto di inizio della lettura) dall'inizio del valore LOB.
4. Occorrenza della sequenza all'interno del valore LOB.

La funzione INSTR all'interno di DBMS\_LOB cerca nel LOB una sequenza specifica di byte o caratteri. La variabile relativa all'occorrenza consente di specificare quale occorrenza della sequenza debba essere restituita all'interno del valore cercato. L'output della funzione INSTR è la posizione iniziale della sequenza all'interno della stringa esaminata. Per esempio, in SQL,

```
INSTR('ABCABC', 'A', 1, 1) = 1
```

significa che all'interno della stringa 'ABCABC' viene cercato il carattere 'A', partendo dalla prima posizione della stringa e cercando la prima occorrenza. 'A' viene trovata alla prima posizione della stringa esaminata. Iniziando la ricerca dalla seconda posizione della stringa, il risultato sarebbe stato il seguente:

```
INSTR('ABCABC', 'A', 2, 1) = 4
```

Dato che questa funzione INSTR inizia dalla seconda posizione, la prima 'A' non fa parte del valore analizzato. Al contrario, la prima 'A' trovata è quella in quarta posizione.

Se si modifica l'ultimo parametro nella funzione SQL INSTR per cercare la seconda occorrenza della stringa 'A', partendo dalla seconda posizione, non viene trovato nulla:

```
INSTR('ABCABC', 'A', 2, 2) = 0
```

Se la funzione SQL INSTR non riesce a trovare l'occorrenza corrispondente della sequenza, viene restituito 0. La funzione INSTR del package DBMS\_LOB funziona nello stesso modo.

Dato che INSTR è una funzione, la sua variabile di output dovrà essere assegnata nello stesso modo in cui è stata assegnata la variabile di output della funzione SUBSTR. Nel listato seguente, viene analizzato il valore CLOB Testo\_Proposta alla ricerca della stringa 'pr'. Per contenere l'output della funzione INSTR viene dichiarata una variabile "var\_posizione".

```
declare
    var_locator    CLOB;
    var_sequenza   VARCHAR2(2);
    var_offset     INTEGER;
    var_occor      INTEGER;
    var_posizione  INTEGER;

begin
    var_sequenza := 'pr';
    var_offset := 1;
    var_occor := 1;
    select Testo_Proposta into var_locator
        from PROPOSTA
       where Proposta_ID = 1;
    var_posizione := DBMS_LOB.INSTR(var_locator, var_sequenza, var_offset, var_occor);
    DBMS_OUTPUT.PUT_LINE('Trovata stringa alla posizione: ' || var_posizione);
end;
```

L'output è il seguente.

```
Trovata stringa alla posizione: 23
```

```
PL/SQL procedure successfully completed.
```

L'output mostra che la stringa cercata 'pr' è stata trovata all'interno di Testo\_Proposta, a partire dal ventitreesimo carattere del valore LOB. Dato che Oracle9i supporta le funzioni di stringa SQL sulle colonne CLOB, questa funzione di chiamata equivale alla query seguente:

```
select INSTR(Testo_Proposta , 'pr', 1, 1)
  from PROPOSTA
 where Proposta_ID = 1;
```

## **GETLENGTH**

La funzione GETLENGTH del package DBMS\_LOB restituisce la lunghezza del valore LOB. Come funzionamento, DBMS\_LOB.GETLENGTH è simile alla funzione SQL LENGTH, ma viene utilizzata solo per i valori LOB.

La funzione GETLENGTH del package DBMS\_LOB possiede solo un parametro di input: il valore del locator per il LOB. Nel listato seguente, vengono dichiarate delle variabili per contenere il valore del locator e il valore di output. La funzione GETLENGTH viene quindi eseguita e il risultato visualizzato tramite la procedura PUT\_LINE:

```

declare
  var_locator CLOB;
  var_lunghezza INTEGER;
begin
  select Testo_Proposta into var_locator
    from PROPOSTA
   where Proposta_ID = 1;
  var_lunghezza := DBMS_LOB.GETLENGTH(var_locator);
  DBMS_OUTPUT.PUT_LINE('Lunghezza di LOB: ' || var_lunghezza);
end;

```

Ecco l'output prodotto da GETLENGTH:

Lunghezza di LOB: 61

PL/SQL procedure successfully completed.

Se il valore LOB è NULL, la funzione GETLENGTH restituisce un valore NULL.

La funzione SQL equivalente per i valori CLOB è

```

select LENGTH(Testo_Proposta)
      from PROPOSTA
     where Proposta_ID = 1;

```

## COMPARE

La funzione COMPARE del package DBMS\_LOB confronta due valori LOB. Se i due valori LOB sono identici, la funzione COMPARE restituisce 0; altrimenti restituisce un valore intero diverso da zero (in genere 1 o -1). Per eseguire la funzione COMPARE, il package DBMS\_LOB esegue essenzialmente due funzioni READ e confronta i risultati. Pertanto, per ciascuno dei valori LOB da confrontare si dovrà fornire un valore del locator e uno di offset. Il numero di caratteri o byte da confrontare è lo stesso per entrambi i valori LOB. Si possono confrontare soltanto valori LOB dello stesso tipo di dati.

La funzione COMPARE prevede cinque parametri di input, che devono essere specificati in questo ordine:

1. Locator LOB per il primo LOB.
2. Locator LOB per il secondo LOB.
3. Variabile di quantità (ovvero, il numero di byte o caratteri da confrontare).
4. Offset (punto di inizio della lettura) dall'inizio del primo valore LOB.
5. Offset (punto di inizio della lettura) dall'inizio del secondo valore LOB.

Dal momento che i due LOB messi a confronto possono avere valori di offset differenti, è possibile confrontare la prima parte di un valore LOB con la seconda parte di un valore LOB differente. L'esempio del listato seguente confronta i primi 25 caratteri dei valori Testo\_Proposta tratti da due voci: il record Proposta\_ID 1 e il record Proposta\_ID 3.

```

declare
  prima_var_locator CLOB;
  seconda_var_locator CLOB;
  var_quantita INTEGER;
  prima_var_offset INTEGER;
  seconda_var_offset INTEGER;
  var_output INTEGER;
begin

```

```

var_quantita      := 25;
prima_var_offset  := 1;
seconda_var_offset := 1;
select Testo_Proposta into prima_var_locator
  from PROPOSTA
 where Proposta_ID = 1;
select Testo_Proposta into seconda_var_locator
  from PROPOSTA
 where Proposta_ID = 3;
var_output := DBMS_LOB.COMPARE(prima_var_locator, seconda_var_locator,
                                var_quantita, prima_var_offset, seconda_var_offset);
DBMS_OUTPUT.PUT_LINE('Valore del confronto (0 se uguali): ' || var_output);
end;

```

Ecco l'output:

```
Valore del confronto (0 se uguali): 1
```

```
PL/SQL procedure successfully completed.
```

Se le due stringhe sono identiche, la funzione COMPARE restituisce 0.

Nel listato seguente, vengono confrontati gli stessi due LOB, questa volta però solo i primi cinque caratteri sono coinvolti nel confronto. Poiché entrambi iniziano con i caratteri ‘Quest’, la funzione COMPARE restituisce 0.

```

declare
  prima_var_locator  CLOB;
  seconda_var_locator CLOB;
  var_quantita        INTEGER;
  prima_var_offset    INTEGER;
  seconda_var_offset  INTEGER;
  var_output          INTEGER;
begin
  var_quantita      := 5;
  prima_var_offset  := 1;
  seconda_var_offset := 1;
  select Testo_Proposta into prima_var_locator
    from PROPOSTA
   where Proposta_ID = 1;
  select Testo_Proposta into seconda_var_locator
    from PROPOSTA
   where Proposta_ID = 3;
  var_output := DBMS_LOB.COMPARE(prima_var_locator, seconda_var_locator,
                                var_quantita, prima_var_offset, seconda_var_offset);
  DBMS_OUTPUT.PUT_LINE('Valore del confronto (0 se uguali): ' || var_output);
end;

```

```
Valore del confronto (0 se uguali): 0
```

```
PL/SQL procedure successfully completed.
```

La funzione COMPARE consente di confrontare stringhe di caratteri con altre stringhe di caratteri e dati binari con altri dati binari. Se si usa la funzione COMPARE su colonne BLOB, è necessario definire le variabili dei locator come tipi di dati RAW.

## WRITE

La procedura WRITE del package DBMS\_LOB consente di scrivere dati in posizioni specifiche del LOB. Per esempio, si possono scrivere dati binari in una sezione di un BLOB, sovrascrivendo i dati preesistenti in tale posizione. Con la procedura WRITE è possibile scrivere dati carattere nei campi CLOB. Questa procedura prevede quattro parametri di input, che devono essere specificati in questo ordine:

1. Locator LOB per il LOB.
2. Variabile di quantità (ovvero il numero di byte o caratteri da scrivere).
3. Offset (punto iniziale di scrittura) dall'inizio del valore LOB.
4. Variabile buffer assegnata alla stringa di caratteri o ai dati binari da aggiungere.

Dato che la procedura WRITE aggiorna il valore LOB, per prima cosa si dovrebbe bloccare la riga tramite un comando select for update, come mostrato nei passaggi scritti in grassetto nel listato seguente. In questo esempio viene selezionato e bloccato un record. Quindi, il testo 'AGGIUNGI NUOVO TESTO' viene scritto all'interno del valore LOB, sostituendo i dati a partire dalla posizione 10 (come definito dalla variabile di offset).

```
declare
    var_locator    CLOB;
    var_quantita   INTEGER;
    var_offset     INTEGER;
    var_buffer     VARCHAR2(12);
begin
    var_quantita := 12;
    var_offset := 10;
    var_buffer := 'AGGIUNGI NUOVO TESTO';
    select Testo_Proposta into var_locator
        from PROPOSTA
       where Proposta_ID = 3
         for update;
    DBMS_LOB.WRITE(var_locator, var_quantita, var_offset, var_buffer);
commit;
end;
```

Dato che la procedura WRITE modifica i dati, occorre aggiungere un comando commit prima della clausola end del blocco PL/SQL.

Per la procedura WRITE non viene fornito alcun output. Per visualizzare i dati modificati, è necessario chiamare la procedura READ (all'interno dello stesso blocco PL/SQL, se lo si desidera) oppure selezionare nuovamente il valore LOB dalla tabella:

```
select Testo_Proposta
      from PROPOSTA
     where Proposta_ID = 3;

TESTO_PROPOSTA
-----
Questo è AGGIUNGI TESTOa proposta.
```

La procedura WRITE ha sovrascritto 12 caratteri, a partire dal decimo carattere di Testo\_Proposta. Per aggiungere dei nuovi dati alla fine di un valore LOB già esistente si può utilizzare la procedura WRITEAPPEND.

Per i dati CLOB, si può riscrivere il tutto con un'istruzione update associata a SUBSTR e alle funzioni di concatenazione.

## APPEND

La procedura APPEND del package DBMS\_LOB aggiunge i dati di un LOB a un secondo LOB. Dato che questa operazione corrisponde all'aggiornamento di un valore LOB, il record da aggiornare dovrebbe essere bloccato prima di eseguire la procedura APPEND.

La procedura APPEND accetta solo due parametri: il valore del locator per il LOB di destinazione e il valore del locator per il LOB sorgente. Se ambedue i parametri sono NULL, la procedura APPEND restituisce un errore.

Nell'esempio seguente vengono definite due variabili per il locator. La prima variabile del locator, "var\_locator\_dest", rappresenta il locator del LOB a cui aggiungere i dati; la seconda, "var\_locator\_sorg", rappresenta il locator dei dati di origine da aggiungere al LOB di destinazione. Quest'ultimo, dovendo essere aggiornato, viene bloccato tramite un comando select for update. Nell'esempio seguente, il valore Testo\_Proposta per il record Proposta\_ID 1 viene aggiunto al valore Testo\_Proposta per il record Proposta\_ID 3:

```
declare
    var_locator_dest    CLOB;
    var_locator_sorg    CLOB;
begin
    select Testo_Proposta into var_locator_dest
        from PROPOSTA
       where Proposta_ID = 3
         for update;
    select Testo_Proposta into var_locator_sorg
        from PROPOSTA
       where Proposta_ID = 1;
    DBMS_LOB.APPEND(var_locator_dest, var_locator_sorg);
commit;
end;
```

Per visualizzare i dati modificati, si selezioni il record dalla tabella PROPOSTA:

```
select Testo_Proposta
      from PROPOSTA
     where Proposta_ID = 3;
```

TESTO\_PROPOSTA

-----  
Questo è AGGIUNGI TEST0a proposta.Questo è il testo di una proposta per  
mantenere un giard

Cosa è successo al resto del testo? Per default, durante le query vengono visualizzati soltanto 80 caratteri dei valori LONG e LOB. Per visualizzare il testo restante, occorre utilizzare il comando set long:

```
set long 1000
```

```
select Testo_Proposta
      from PROPOSTA
     where Proposta_ID = 3;
```

TESTO\_PROPOSTA

-----  
Questo è AGGIUNGI TEST0a proposta.Questo è il testo di una proposta per  
mantenere un giardino organico

In questo esempio, vengono visualizzati i primi 1000 caratteri del valore LOB. Per sapere quanto è lungo il LOB, occorre utilizzare la funzione GETLENGTH nel modo descritto in precedenza nel capitolo.

Oltre a consentire di aggiungere valori CLOB, la procedura APPEND consente anche di aggiungere valori BLOB. Per aggiungere dati di tipo BLOB, è necessario conoscere il formato dei dati stessi e l'effetto dell'aggiunta di nuovi dati sulla fine del valore BLOB. Per i valori BLOB la procedura APPEND può essere utile nel caso di applicazioni in cui l'oggetto BLOB contiene solo tipi di dati RAW (come quelli utilizzati da applicazioni di trasmissione digitale dei dati) al posto di dati binari formattati (come file di fogli di calcolo formattati da un programma).

Per i dati CLOB, si potrebbe riscrivere il tutto con un'istruzione update, impostando il valore Testo\_Proposta uguale alla concatenazione di più valori.

## **ERASE**

Per cancellare i caratteri o i byte in qualsiasi punto di un LOB, è possibile ricorrere alla procedura ERASE del package DBMS\_LOB. Questa stessa procedura può essere utilizzata anche per cancellare l'intero valore LOB, oppure si può specificare quale parte del LOB cancellare. Come funzionamento, la procedura ERASE è simile alla procedura WRITE. Se si eliminano dei dati in un valore BLOB, questi vengono sostituiti da caratteri di riempimento a zero byte. Se si eliminano dei dati in un valore CLOB, al loro posto vengono inseriti degli spazi.

Dato che la procedura comporta l'aggiornamento di un valore LOB, è necessario seguire le procedure standard per gli aggiornamenti LOB: blocco della riga e commit al termine della procedura.

Nel blocco PL/SQL seguente viene cancellata una parte di Testo\_Proposta per il record con Proposta\_ID 3. I caratteri cancellati partiranno da un offset pari a 10 e continueranno per 17 caratteri. I parametri della procedura ERASE sono, nell'ordine:

1. Locator LOB per il LOB.
2. Variabile di quantità (ovvero, il numero di byte o caratteri da eliminare).
3. Offset (punto iniziale di cancellazione) dall'inizio del valore LOB.

```
declare
  var_locator CLOB;
  var_quantita INTEGER;
  var_offset   INTEGER;
begin
  var_quantita := 17;
  var_offset := 10;
  select TestoProposta into var_locator
    from PROPOSTA
   where Proposta_ID = 3
     for update;
  DBMS_LOB.ERASE(var_locator, var_quantita, var_offset);
commit;
end;
```

Per visualizzare la modifica al record è sufficiente eseguire una query del valore Testo\_Proposta:

```
select Testo_Proposta
  from PROPOSTA
 where Proposta_ID = 3;
```

**TESTO\_PROPOSTA**

Questo è t esto.Questo è il testo di una proposta per mantenere un giardino organico.

**NOTA** *Lo spazio precedentemente occupato dai dati cancellati viene riempito con spazi vuoti. La posizione dei restanti dati del LOB non cambia.*

Per i dati CLOB è possibile riscrivere questa operazione con un'istruzione update, impostando un nuovo valore per Testo\_Proposta con la concatenazione dei risultati di SUBSTR e una stringa vuota.

**TRIM**

La procedura TRIM del package DBMS\_LOB consente di ridurre la dimensione di un valore LOB eliminando caratteri o byte dalle estremità del valore (come la funzione SQL RTRIM). Per eseguire la procedura TRIM è necessario specificare il valore del locator per il LOB e la nuova lunghezza del LOB.

Nel listato seguente, vengono rimossi i primi dieci caratteri di Testo\_Proposta per il record Proposta\_ID 3. Dato che la procedura WRITE modifica i dati, si aggiunga un comando commit prima della clausola end del blocco PL/SQL.

```
declare
  var_locator      CLOB;
  nuova_var_lung  INTEGER;
begin
  nuova_var_lung   := 10;
  select Testo_Proposta into var_locator
    from PROPOSTA
   where Proposta_ID = 3
     for update;
  DBMS_LOB.TRIM(var_locator, nuova_var_lung);
commit;
end;
```

Per la procedura TRIM non viene fornito alcun output. Per visualizzare i dati modificati, occorre selezionare nuovamente il valore LOB dalla tabella:

```
select Testo_Proposta
  from PROPOSTA
 where Proposta_ID = 3;
```

**TESTO\_PROPOSTA**

Questo è i

L'output di esempio mostra i risultati della procedura TRIM successiva alla procedura ERASE eseguita nel paragrafo precedente. Per i dati CLOB, il tutto può essere riscritto come una semplice istruzione update:

```
update PROPOSTA
  set Testo_Proposta = SUBSTR(Testo_Proposta,1,10)
 where Proposta_ID = 3;
```

## COPY

La procedura COPY del package DBMS\_LOB consente di copiare i dati da una posizione di un LOB in un secondo LOB. A differenza di quanto avviene con la procedura APPEND, non è necessario copiare l'intero testo di un valore LOB in un altro. Durante la procedura COPY, è possibile specificare l'offset di lettura e l'offset di scrittura. I cinque parametri della procedura COPY sono, nell'ordine:

1. Locator del LOB di destinazione.
2. Locator del LOB sorgente.
3. Variabile di quantità (ovvero, il numero di caratteri o byte da copiare).
4. Offset di inizio della scrittura all'interno del valore LOB di destinazione.
5. Offset di inizio della lettura all'interno del valore LOB di destinazione.

La procedura COPY combina le capacità delle procedure READ e WRITE. Come avviene per l'operazione WRITE, la procedura COPY modifica il valore LOB. Per questo motivo, sarebbe opportuno bloccare prima il record. Inoltre il blocco PL/SQL deve contenere un comando commit. Nell'esempio seguente, i primi 61 caratteri del record Proposta\_ID 1 vengono copiati nel record Proposta\_ID 3. Dato che il LOB di destinazione utilizza un offset pari a 1, i dati copiati *sovrascrivono* i dati già presenti nel LOB di destinazione.

```
declare
    var_locator_dest    CLOB;
    var_locator_sorg    CLOB;
    var_quantita        INTEGER;
    var_offset_dest     INTEGER;
    var_offset_sorg     INTEGER;
begin
    var_quantita      := 61;
    var_offset_dest   := 1;
    var_offset_sorg   := 1;
    select Testo_Proposta into var_locator_dest
        from PROPOSTA
       where Proposta_ID = 3
         for update;
    select Testo_Proposta into var_locator_sorg
        from PROPOSTA
       where Proposta_ID = 1;
    DBMS_LOB.COPY(var_locator_dest, var_locator_sorg,
                  var_quantita, var_offset_dest, var_offset_sorg);
commit;
end;
```

La procedura COPY non visualizza output. Per visualizzare i dati modificati occorre selezionare nuovamente il valore LOB dalla tabella:

```
select Testo_Proposta
  from PROPOSTA
 where Proposta_ID = 3;
```

```
TESTO_PROPOSTA
```

---

Questo è il testo di una proposta per mantenere un giardino organico.

## Uso di funzioni e procedure per BFILE

Dato che i valori BFILE sono memorizzati esternamente, esistono numerose funzioni e procedure utilizzate per manipolare i file prima di eseguire operazioni READ, SUBSTR, INSTR, GETLENGTH o COMPARE su LOB BFILE. Le procedure e funzioni relative ai valori BFILE all'interno del package DBMS\_LOB, elencate precedentemente nella Tabella 32.1, consentono di aprire file, chiudere file, ottenere il nome del file, verificare l'esistenza di un file e controllare se un file è aperto. La funzione per il controllo dell'esistenza è necessaria, perché Oracle non gestisce i dati memorizzati nel file esterno, ma solo il puntatore creato mediante la procedura BFILENAME al momento dell'inserimento o dell'aggiornamento del record. Per controllare i file utilizzati dal codice PL/SQL, si dovrebbero usare le procedure e le funzioni elencate nella Tabella 32.1. Per esempio, per leggere i primi dieci byte dal BFILE Lettera\_Present per il record Proposta\_ID 2, si dovrebbe aprire il file ed eseguire la procedura READ. Quindi, i dati letti potranno essere visualizzati mediante la procedura PUT\_LINE del package DBMS\_OUTPUT.

Se nel blocco PL/SQL si utilizza una sezione Gestione delle eccezioni, sarà necessario includere chiamate alla procedura FILECLOSE all'interno delle eccezioni dichiarate. Per ulteriori informazioni sulla gestione delle eccezioni all'interno dei blocchi PL/SQL, si rimanda al Capitolo 27.

Per spostare i dati da un file esterno in un tipo di dati BLOB, è possibile creare un tipo di dati BFILE in una tabella e implementare la funzione BFILENAME per creare un locator LOB che punta al file esterno. La funzione BFILENAME accetta due parametri: il nome della directory e il nome del file. Per esempio:

```
insert into PROPOSTA (Proposta_ID, Lettera_Prsent)
values (4, BFILENAME (' dir_proposta_ ','extfile.doc'));
```

Ora si può aprire il file e usare la procedura LOADFROMFILE per caricare il tipo di dati BLOB insieme ai dati presi dal file esterno.

Se attualmente i dati sono nel tipo LONG, li si può modificare in una colonna di tipo CLOB o NCLOB tramite il comando alter table. Per trasformare una colonna di tipo LONG RAW in una di tipo BLOB si può utilizzare alter table. Per le operazioni insert as select, si può ricorrere alla funzione SQL TO\_LOB per spostare i dati dalla colonna di tipo LONG in una colona BLOB.

## Cancellazione dei LOB

Quando si eliminano valori LOB, viene cancellato anche il valore del locator. Se il valore eliminato è un LOB interno (BLOB o CLOB o NCLOB), vengono cancellati sia il valore del locator sia il valore LOB. Se invece il valore eliminato è un LOB esterno (BFILE), viene cancellato solo il valore del locator, mentre il file cui punta il locator dovrà essere eliminato manualmente.



## Capitolo 33

# Concetti avanzati di orientamento agli oggetti

- 33.1 **Oggetti riga e oggetti colonna**
- 33.2 **Tabelle oggetto e identificativi di oggetto (OID)**
- 33.3 **Viste oggetto con REF**
- 33.4 **PL/SQL a oggetti**
- 33.5 **Oggetti nel database**

**F**inora tutte le caratteristiche di programmazione a oggetti di Oracle descritte in questo libro condividono due caratteristiche: la qualità di oggetti incorporati e la qualità di oggetti colonna. Un *oggetto incorporato* è completamente contenuto all'interno di un altro oggetto. Per esempio, una tabella annidata è contenuta da un'altra tabella, quindi è un oggetto incorporato. Anche se i dati di una tabella annidata vengono memorizzati separatamente dalla tabella principale, è possibile accedervi solo attraverso la tabella principale. Un *oggetto colonna* viene rappresentato da una colonna di una tabella. Per esempio, un array variabile è rappresentato da una colonna di una tabella, quindi è un oggetto colonna.

Per sfruttare le caratteristiche della programmazione a oggetti, un database deve supportare anche *oggetti riga*, oggetti rappresentati come righe invece che come colonne. Gli oggetti riga non sono oggetti incorporati, bensì *oggetti referenziali*, accessibili tramite riferimenti da altri oggetti. In questo capitolo viene spiegato come creare e fare riferimento a oggetti riga.

Come si vedrà, gli oggetti riga possono essere estesi in modo da essere usati con alcuni degli oggetti descritti nei capitoli precedenti (per esempio le viste oggetto). Oltre agli oggetti riga, viene trattata anche l'applicazione degli oggetti al linguaggio PL/SQL, ovvero la creazione di *oggetti PL/SQL*.

## 33.1 Oggetti riga e oggetti colonna

La distinzione tra oggetti riga e oggetti colonna è fondamentale. Per prima cosa si considerino gli oggetti colonna. Questi oggetti si basano su estensioni di caratteristiche già presenti nel database. Un tipo di dati astratto o un array variabile possono essere incorporati come una colonna all'interno di una tabella. Una tabella annidata o un oggetto di grandi dimensioni (LOB) possono essere memorizzati in un'unica colonna di una tabella. In ogni caso, l'oggetto viene rappresentato come una colonna all'interno di una tabella.

Le tabelle annidate e i LOB comportano la memorizzazione di dati fuori linea rispetto alla tabella principale. Quando si usa un LOB (Capitolo 32) è necessario specificare i valori di memorizzazione per la tabella che memorizzerà i dati LOB. Oracle crea dei locator LOB che puntano dalla tabella principale alle righe della tabella LOB. Quando si crea una tabella annidata, è necessario specificare il nome della tabella in cui i record della tabella annidata vengono me-

memorizzati. Oracle crea puntatori dalla tabella principale ai record nelle tabelle annidate. In questo modo, la tabella principale e la sua tabella annidata incorporata sono correlate.

Negli oggetti riga, gli oggetti vengono rappresentati come righe e non come colonne. I dati non sono incorporati nella tabella principale, che invece contiene un riferimento a un'altra tabella. Le tabelle annidate, in quanto memorizzate separatamente dalla tabella principale, mettono a disposizione uno strumento utile per comprendere gli oggetti riga. Che cosa accade se i dati della tabella annidata non sono incorporati all'interno della tabella principale e ciascuna delle righe di questa tabella è un oggetto riga? La tabella principale definirà riferimenti ai dati correlati.

Le caratteristiche relazionali a oggetti di Oracle descritte nei primi capitoli si basavano su oggetti colonna. In questo capitolo, l'obiettivo è puntato sulle capacità di OOP avanzate, basate sugli oggetti riga. Gli oggetti riga vengono utilizzati per creare riferimenti tra righe di tabelle differenti.

### 33.2 Tabelle oggetto e identificativi di oggetto (OID)

In una *tabella oggetto* ciascuna riga è un oggetto riga. Una tabella oggetto differisce per molti aspetti da una normale tabella relazionale. In primo luogo, ogni riga all'interno della tabella oggetto ha un valore di identificazione oggetto (OID, Object IDentifier) assegnato da Oracle quando viene creata. In secondo luogo, alle righe di una tabella oggetto possono fare riferimento altri oggetti all'interno del database.

Per creare una tabella oggetto si utilizzi il comando `create table`. Si consideri il tipo di dati `ANIMALE_TY` descritto nei capitoli precedenti:

```
create or replace type ANIMALE_TY as object
(Famiglia    VARCHAR2(25),
 Nome        VARCHAR2(25),
 DataNascita DATE);
```

**NOTA** Per mantenere l'esempio semplice, il tipo di dati `ANIMALE_TY` viene creato senza alcuna funzione membro.

Per creare una tabella oggetto del tipo di dati `ANIMALE_TY` si esegua il comando `create table` riportato di seguito:

```
create table ANIMALE of ANIMALE_TY;
```

Table created.

Si osservi che il comando ha una sintassi inusuale, in quanto crea la tabella “di” un tipo di dati astratto. La tabella così ottenuta può apparire inizialmente come una normale tabella relazionale:

```
describe ANIMALE
```

Nome	Null?	Type
FAMIGLIA		VARCHAR2(25)
NOME		VARCHAR2(25)
DATANASCITA		DATE

Le colonne della tabella ANIMALE sono associate agli attributi del tipo di dati ANIMALE\_TY. Esistono però differenze significative nel modo in cui si può utilizzare la tabella, in quanto si tratta di una tabella oggetto. Come si è notato in precedenza nel paragrafo, ciascuna riga all'interno della tabella oggetto ha un valore OID, ed è possibile fare riferimento alle righe come a oggetti.

## Inserimento di righe nelle tabelle oggetto

Dato che ANIMALE è una tabella oggetto, è possibile utilizzare il constructor method del suo tipo di dati quando si inseriscono i dati nella tabella. Dal momento che la tabella ANIMALE si basa sul tipo di dati astratto ANIMALE\_TY, il constructor method ANIMALE\_TY potrà essere utilizzato quando si inseriscono dei valori in questa tabella.

**NOTA** *Per delle analisi dettagliate dei constructor method, si rimanda ai Capitoli 4 e 30.*

Nel listato seguente vengono inserite tre righe nella tabella oggetto ANIMALE. In ogni comando viene chiamato il constructor method ANIMALE\_TY.

```
insert into ANIMALE values
  (ANIMALE_TY('MULO', 'FRANCES', '01-APR-02'));
insert into ANIMAL values
  (ANIMALE_TY('CANE', 'BENJI', '03-SEP-01'));
insert into ANIMAL values
  (ANIMALE_TY('COCCODRILLO', 'LYLE', '14-MAY-00'));
```

Se il tipo di dati astratto su cui è basata la tabella oggetto si basa a sua volta su un secondo tipo di dati astratto, potrebbe essere necessario annidare le chiamate a più constructor method all'interno del comando insert (per esempi di tipi di dati astratti annidati si rimanda al Capitolo 4).

**NOTA** *Se una tabella oggetto si basa su un tipo di dati che non contiene tipi di dati annidati, è possibile inserire dei record nella tabella oggetto utilizzando per il comando insert la normale sintassi delle tabelle relazionali.*

Quando si inserisce una riga in una tabella oggetto, Oracle assegna alla riga un valore OID. La disponibilità di un OID consente di utilizzare la riga come un oggetto cui è possibile fare riferimento. I valori OID non vengono riutilizzati.

## Selezione di valori dalle tabelle oggetto

Quando è stata creata, la tabella ANIMALE si basava interamente sul tipo di dati ANIMALE\_TY:

```
create table ANIMALE of ANIMALE_TY;
```

Quando si esegue una selezione da una tabella che contiene un tipo di dati astratto, si fa riferimento alle colonne del tipo di dati astratto come a parti delle colonne della tabella. In altre parole, se si utilizza il tipo di dati ANIMALE\_TY come base per una *colonna* di nome Animale, si potranno selezionare i nomi degli animali selezionando Animale.Nome dalla tabella.

Una tabella oggetto si basa su un tipo di dati e non ha altre colonne. Per questo motivo, non è necessario fare riferimento al tipo di dati quando si accede alle colonne. Per selezionare i nomi dalla tabella ANIMALE, è necessario eseguire una query diretta sugli attributi del tipo di dati:

```
select Nome
  from ANIMALE;
```

NOME

```
-----
FRANCES
BENJI
LYLE
```

È possibile fare riferimento alle colonne all'interno della clausola `where` come se `ANIMALE` fosse una tabella relazionale:

```
select Nome
  from ANIMALE
 where Famiglia = 'COCCODRILLO';
```

NOME

```
-----
LYLE
```

Se il tipo di dati `ANIMALE_TY` utilizzasse un altro tipo di dati astratto per una delle sue colonne, si dovrebbe fare riferimento alla colonna di questo tipo durante tutti gli inserimenti, aggiornamenti e cancellazioni. Per esempi di query da tipi di dati astratti annidati, si rimanda al Capitolo 4.

## Aggiornamenti e cancellazioni da tabelle oggetto

Se la tabella oggetto si basa su un tipo di dati astratto che non utilizza altri tipi di dati astratti, l'aggiornamento o la cancellazione di una tabella oggetto usano lo stesso formato che si adopererebbe per le tabelle relazionali. In questo esempio, il tipo di dati `ANIMALE_TY` non utilizza altri tipi di dati astratti per nessuna delle sue colonne. Pertanto, per gli aggiornamenti e le cancellazioni dalla tabella oggetto `ANIMALE` ci si comporta come se questa fosse una tabella relazionale.

Nel comando seguente, viene aggiornata una delle righe di `ANIMALE`:

```
update ANIMALE
  set DataNascita = '01-MAY-00'
 where Nome = 'LYLE';
```

1 row updated.

Si osservi che durante l'aggiornamento si fa riferimento alle colonne della tabella oggetto come se `ANIMALE` fosse una tabella relazionale. Durante una cancellazione, è possibile utilizzare gli stessi mezzi per fare riferimento alle colonne della tabella oggetto nella clausola `where`:

```
delete from ANIMALE
 where Nome = 'LYLE';
```

1 row deleted.

D'altra parte, tenere un coccodrillo insieme ai muli e ai cani non era una buona idea.

## La funzione REF

La funzione REF consente di fare riferimento a oggetti riga già esistenti. Per esempio, la tabella ANIMALE (dopo la cancellazione di una riga, descritta nel paragrafo precedente) ha due oggetti riga, a ciascuno dei quali è stato assegnato un valore OID. È possibile visualizzare gli OID assegnati a ciascuna riga tramite la funzione REF, come mostrato di seguito:

```
select REF(A)
  from ANIMALE A
 where Nome = 'FRANCES';

REF(A)
-----
0000280209FFD8EC317DA111D6B00844553544200FFD8EC307
DA111D6B00844553544200004123160000
```

**NOTA** *Il valore REF dovrebbe essere diverso da quello mostrato in questo esempio, ma dovrebbe presentare lo stesso formato.*

In questo esempio, alla tabella ANIMALE è stato assegnato l'alias “A”, che è stato utilizzato come input per la funzione REF. L'output mostra il valore OID dell'oggetto riga che ha soddisfatto la condizione di limitazione della query.

**NOTA** *La funzione REF accetta come input l'alias assegnato alla tabella oggetto. Anche le altre funzioni, descritte più avanti in questo capitolo, accettano gli alias come input, quindi si consiglia di familiarizzare con la sintassi descritta.*

Dato che la funzione REF è in grado di fare riferimento solo a oggetti riga, non è possibile servirsene per fare riferimento a oggetti colonna. Questi ultimi comprendono tipi di dati astratti, LOB e collezionatori.

Come si può vedere, la funzione REF di per sé non fornisce alcuna informazione utile. È necessario utilizzare un'altra funzione, DEREF, che converte l'output di REF in valori. Sarà quindi possibile utilizzare REF e DEREF per fare riferimento a valori di oggetti riga, come descritto nel prossimo paragrafo.

## Uso della funzione DEREF

La funzione REF accetta un oggetto riga come argomento e restituisce un valore di riferimento. La funzione DEREF esegue l'operazione opposta, ossia accetta un valore di riferimento (l'OID creato per un riferimento) e restituisce il valore dell'oggetto riga.

In precedenza nel capitolo è stata creata la tabella oggetto ANIMALE con il tipo di dati astratto ANIMALE\_TY:

```
create table ANIMALE of ANIMALE_TY;
```

Per comprendere come DEREF e REF lavorino insieme, si consideri una tabella correlata alla tabella ANIMALE. La tabella PASTORE terrà traccia delle persone che si prendono cura degli animali. Questa tabella conterrà una colonna per il nome del pastore (NomePastore) e un riferimento alla tabella oggetto ANIMALE (AnimaleCurato), come mostrato nel listato seguente:

```
create table PASTORE
(NomePastore      VARCHAR2(25),
 AnimaleCurato    REF ANIMALE_TY);
```

La prima parte del comando `create table` ha un aspetto normale:

```
create table PASTORE
(NomePastore      VARCHAR2(25),
```

ma l'ultima linea mostra una nuova caratteristica:

```
AnimaleCurato    REF ANIMALE_TY);
```

La colonna `AnimaleCurato` fa riferimento a dati memorizzati altrove. La funzione `REF` punta la colonna `AnimaleCurato` a un oggetto riga del tipo di dati `ANIMALE_TY`. Dato che `ANIMALE` è una tabella oggetto del tipo di dati `ANIMALE_TY`, la colonna `AnimaleCurato` può puntare agli oggetti riga all'interno della tabella oggetto `ANIMALE`. La funzione `REF` può puntare a qualsiasi tabella oggetto di questo tipo.

Quando si descrive la tabella `PASTORE`, si può notare che la sua colonna `AnimaleCurato` si basa su un `REF`:

```
describe PASTORE
```

Nome	Null?	Type
NOME_PASTORE		VARCHAR2(25)
ANIMALECURATO		REF OF ANIMALE_TY

Per visualizzare tutti i dettagli, si utilizzi un valore maggiore per la profondità di descrizione:

```
set describe depth 2
```

```
desc PASTORE
Nome           Null?   Type
-----
NOME_PASTORE          VARCHAR2(25)
ANIMALECURATO        REF OF ANIMALE_TY
FAMIGLIA            VARCHAR2(25)
NOME                VARCHAR2(25)
DATANASCITA         DATE
```

Ora, si inserisca un record in `PASTORE`. Per memorizzare i riferimenti `ANIMALE` nella colonna `AnimaleCurato`, è necessario il ricorso alla funzione `REF`.

```
insert into PASTORE
select 'CATHERINE WEILZ',
       REF(A)
     from ANIMALE A
   where Nome = 'BENJI';
```

Si osservi attentamente il comando `insert`. Per prima cosa, il valore `NomePastore` viene selezionato come valore letterale:

```
insert into PASTORE
select 'CATHERINE WEILZ'.
```

In seguito, si deve specificare il valore per la colonna AnimaleCurato. Ma il tipo di dati di AnimaleCurato è REF OF ANIMALE\_TY, quindi per popolare la colonna AnimaleCurato si dovrà selezionare il riferimento dalla tabella oggetto ANIMALE:

```
REF(A)
from ANIMALE A
where Nome = 'BENJI';
```

Cosa è accaduto? Per prima cosa, viene eseguita una query sulla tabella oggetto ANIMALE, e la funzione REF restituisce il valore OID per l'oggetto riga selezionato. Questo valore viene poi memorizzato nella tabella PASTORE come puntatore a questo oggetto riga nella tabella oggetto ANIMALE.

La tabella PASTORE contiene informazioni sugli animali? No: PASTORE contiene il nome del pastore e un riferimento a un oggetto riga nella tabella ANIMALE. Per visualizzare il valore ODI di riferimento, si esegua una query su PASTORE:

```
select * from PASTORE;
NOMEPESTORE
-----
ANIMALECURATO
-----
CATHERINE WEILZ
0000220208FFD8EC327DA111D6B008444553544200FFD8EC307DA111D6B0084445
53544200
```

La colonna AnimaleCurato, come mostra questo listato, contiene il riferimento all'oggetto riga, non il valore dei dati in esso memorizzati.

Non è possibile visualizzare il valore cui si fa riferimento, a meno che non si utilizzi la funzione DEREF. Quando si esegue una selezione da PASTORE, Oracle utilizza il valore OID per valutare il riferimento (REF). La funzione DEREF accetta il riferimento e restituisce un valore.

Nella query seguente, il valore della colonna AnimaleCurato nella tabella PASTORE è il parametro passato alla funzione DEREF. DEREF prende il valore OID da AnimaleCurato e trova l'oggetto cui si riferisce; la funzione valuterà il riferimento e restituirà i valori all'utente.

```
select DEREF(K.AnimaleCurato)
  from PASTORE K
 where NomePastore = 'CATHERINE WEILZ';
DEREF(K.ANIMALECURATO)(FAMIGLIA, NOME, DATANASCITA)
-----
ANIMALE_TY('CANE', 'BENJI', '03-SEP-01')
```

**NOTA** *Il parametro per la funzione DEREF è il nome della colonna REF, non il nome della tabella.*

Il risultato mostra che il record CATHERINE WEILZ in PASTORE fa riferimento a un record di ANIMALE per l'animale di nome 'BENJI.' Alcuni aspetti di questa query sono degni di nota:

- La query utilizza un riferimento a un oggetto riga per passare da una tabella (PASTORE) a una seconda tabella (la tabella oggetto ANIMALE). Pertanto in background viene eseguito un join, senza che vengano specificati criteri per l'operazione.
- La stessa tabella oggetto non viene menzionata nella query. L'unica tabella elencata nella query è PASTORE. Non è necessario conoscere il nome della tabella oggetto per applicare DEREF ai suoi valori.
- Viene restituito l'intero oggetto riga cui si fa riferimento, non solo una parte della riga.

Si tratta di differenze significative che distinguono le query oggetto dalle normali query relazionali. Quindi, durante le query sulle tabelle, è necessario sapere in che modo sono state stabilite le loro relazioni. Queste relazioni si basano su chiavi esterne e chiavi primarie, oppure su tabelle oggetto e tipi di dati REF? Per agevolare il passaggio dall'approccio relazionale a quello orientato agli oggetti, Oracle consente di creare viste oggetto che contengono REF sovrapposti alle tabelle relazionali già esistenti. Si rimanda al paragrafo "Viste oggetto con REF", più avanti nel capitolo, per maggiori dettagli.

## La funzione VALUE

La funzione DEREF è stata applicata a una tabella relazionale, in questo caso la tabella PASTORE. Questa funzione restituisce il valore del riferimento che va dalla tabella relazionale alla tabella oggetto.

Che ne è delle query dalla tabella oggetto? È possibile effettuare selezioni da ANIMALE?

```
select * from ANIMALE;
-----
```

FAMIGLIA	NOME	DATANASCITA
MULO	FRANCES	01-APR-02
CANE	BENJI	03-SEP-01

Anche se ANIMALE è una tabella oggetto, è possibile effettuare delle selezioni da questa tabella, come se si trattasse di una tabella relazionale. Ciò è coerente con gli esempi di inserimenti e cancellazioni descritti in precedenza nel capitolo. Si tratta tuttavia di un'operazione diversa da quella mostrata per mezzo di DEREF. La funzione DEREF visualizzava la struttura completa del tipo di dati astratto utilizzato dalla tabella oggetto ANIMALE:

```
select DEREF(K.AnimaleCurato)
  from PASTORE K
 where NomePastore = 'CATHERINE WEILZ';
-----
```

```
DEREF(K.ANIMALECURATO)(FAMIGLIA, NOME, DATANASCITA)
-----
```

```
ANIMALE_TY('CANE', 'BENJI', '03-SEP-01')
```

Per visualizzare le stesse strutture mediante una query sulla tabella oggetto ANIMALE, occorre utilizzare la funzione VALUE. Come si può vedere nel listato seguente, la funzione VALUE visualizza i dati nello stesso formato utilizzato da DEREF. Il parametro per la funzione VALUE è l'alias della tabella.

```

select VALUE(A)
  from ANIMALE A
 where Nome = 'BENJI';

VALUE(A)(FAMIGLIA, NOME, DATANASCITA)
-----
ANIMALE_TY('CANE', 'BENJI', '03-SEP-01')

ANIMALE_TY('CANE', 'BENJI', '03-SEP-01')

```

La funzione `VALUE` si rivela utile per l'esecuzione del debug di riferimenti e all'interno di PL/SQL, come descritto nel paragrafo “PL/SQL a oggetti”, più avanti in questo capitolo. Dato che questa funzione consente di eseguire query sui valori formattati direttamente dalla tabella oggetto, questi valori possono essere selezionati senza utilizzare la query `DEREF` della colonna `AnimaleCurato` di `PASTORE`.

## Riferimenti non validi

È possibile cancellare l'oggetto cui punta un riferimento. Per esempio, si può eliminare una riga dalla tabella oggetto `ANIMALE` cui punta un record di `PASTORE`:

```

delete from ANIMALE
  where Nome = 'BENJI';

```

Il record di `PASTORE` che fa riferimento a questo record `ANIMALE` avrà un *REF non valido*. Se si inserisce una nuova riga `ANIMALE` per l'animale di nome ‘`BENJI`’, questa non verrà riconosciuta come parte dello stesso riferimento stabilito in precedenza. La spiegazione è la seguente: la prima volta che è stata inserita una riga ‘`BENJI`’, Oracle ha generato un valore `OID` per l'oggetto riga, ed è proprio a questo valore che faceva riferimento la colonna `PASTORE`. Quando l'oggetto riga è stato cancellato, insieme a esso è scomparso anche il valore `OID`, e Oracle non riutilizza i numeri di `OID`. Di conseguenza, quando viene inserito il nuovo record `BENJI`, gli viene assegnato un nuovo valore `OID`, mentre il record di `PASTORE` punta ancora al vecchio valore.

Si tratta di una differenza fondamentale tra i sistemi relazionali e i sistemi a oggetti. In un sistema relazionale, il join di due tabelle dipende solo dai dati correnti. In un sistema orientato agli oggetti, il join avviene tra oggetti: il fatto che due oggetti abbiano gli stessi dati non significa che questi oggetti siano uguali.

### 33.3 Viste oggetto con REF

Per sovrapporre le strutture OOP su tabelle relazionali già esistenti (fare riferimento al Capitolo 30), si può ricorrere alle viste oggetto. Per esempio, si possono creare tipi di dati astratti e usarli all'interno della vista oggetto di una tabella già esistente. L'uso delle viste oggetto consente di accedere alla tabella sia con la sintassi per comandi relazionali, sia con la sintassi per tipi di dati astratti. Per questo motivo, le viste oggetto mettono a disposizione un importante ponte tecnologico che porta dalle applicazioni relazionali esistenti alle applicazioni relazionali ad oggetti.

## Note sulle viste oggetto

L'esempio del Capitolo 30 viene riproposto in questo capitolo come parte delle basi necessarie per viste oggetto avanzate. Per prima cosa, viene creata una tabella CLIENTE con la colonna Cliente\_ID definita come chiave primaria:

```
create table CLIENTE
(Cliente_ID NUMBER constraint CLIENTE_PK primary key,
Nome      VARCHAR2(25),
Via       VARCHAR2(50),
Citta     VARCHAR2(25),
Prov      CHAR(2),
Cap       NUMBER);
```

Successivamente, vengono creati due tipi di dati astratti. Il primo, INDIRIZZO\_TY, contiene attributi per gli indirizzi: via, città, provincia e CAP. Il secondo, PERSONA\_TY, contiene un attributo Nome e un attributo Indirizzo che utilizza il tipo di dati INDIRIZZO\_TY, come mostrato nel listato seguente:

```
create or replace type INDIRIZZO_TY as object
(Via      VARCHAR2(50),
Citta    VARCHAR2(25),
Prov     CHAR(2),
Cap      NUMBER);
create or replace type PERSONA_TY as object
(Nome     VARCHAR2(25),
Indirizzo INDIRIZZO_TY);
```

Dato che la tabella CLIENTE è stata creata senza utilizzare i tipi di dati INDIRIZZO\_TY e PERSONA\_TY, si dovrà ricorrere alle viste oggetto per accedere ai dati di CLIENTE tramite accessi basati su oggetti (come i metodi). È possibile creare una vista oggetto che specifica i tipi di dati astratti che si applicano alla tabella CLIENTE. Nel listato seguente viene creata la vista oggetto CLIENTE\_OV:

```
create view CLIENTE_OV (Cliente_ID, Persona) as
select Cliente_ID,
       PERSONA_TY(Nome,
                  INDIRIZZO_TY(Via, Citta, Prov, Cap))
  from CLIENTE;
```

Durante la creazione della vista oggetto CLIENTE\_OV, vengono specificati i constructor method per i due tipi di dati astratti (INDIRIZZO\_TY e PERSONA\_TY). Ora l'accesso alla tabella CLIENTE può avvenire direttamente (come a una tabella relazionale) o attraverso i constructor method per i tipi di dati astratti.

Nella prossima serie di esempi proposti in questo capitolo viene utilizzata la tabella CLIENTE.

## Viste oggetto che comportano riferimenti

Se la tabella CLIENTE descritta nel paragrafo precedente è correlata a un'altra tabella, le viste oggetto potranno essere utilizzate per creare un riferimento tra le tabelle. In altre parole, Oracle sfrutterà le relazioni chiave primaria/chiave esterna già esistenti per simulare valori OID da utilizzare mediante REF tra le tabelle. Sarà così possibile accedere alle tabelle sia come a tabelle relazionali sia come a oggetti. Quando le tabelle vengono trattate come oggetti, si potrà ricorrere

ai REF per eseguire automaticamente join delle tabelle (per alcuni esempi, si rimanda al paragrafo “Uso della funzione DEREF” più indietro nel capitolo).

La tabella CLIENTE ha una chiave primaria Cliente\_ID. Si crei una tabella piccola che contenga un riferimento di chiave esterna alla colonna Cliente\_ID. Nel listato seguente, viene creata la tabella CLIENTE\_CHIAMATA. La chiave primaria della tabella CLIENTE\_CHIAMATA è la combinazione di Cliente\_ID e Numero\_Chiamata. La colonna Cliente\_ID di CLIENTE\_CHIAMATA è una chiave esterna nei confronti di CLIENTE: non è possibile registrare una chiamata per un cliente che non dispone già di un record in CLIENTE. All’interno della tabella CLIENTE\_CHIAMATA viene creato un singolo attributo non chiave, Data\_Chiamata.

```
create table CLIENTE_CHIAMATA
(Cliente_ID      NUMBER,
 Numero_Chiamata NUMBER,
 Data_Chiamata   DATE,
 constraint CLIENTE_CHIAMATA_PK
     primary key (Cliente_ID, Numero_Chiamata),
 constraint CLIENTE_CHIAMATA_FK foreign key (Cliente_ID)
     references CLIENTE(Cliente_ID));
```

Per esempio, si potrebbero avere le seguenti voci CLIENTE e CLIENTE\_CHIAMATA:

```
insert into CLIENTE values
(123,'SIGMUND','47 HAFFNER RD','LEWISTON','NJ',22222);

insert into CLIENTE values
(234,'EVELYN','555 HIGH ST','LOWLANDS PARK','NE',33333);

insert into CLIENTE_CHIAMATA values
(123,1,TRUNC(SysDate)-1);
insert into CLIENTE_CHIAMATA values
(123,2,TRUNC(SysDate));
```

La chiave esterna da CLIENTE\_CHIAMATA a CLIENTE definisce la relazione tra le tabelle. Da un punto di vista orientato agli oggetti, i record di CLIENTE\_CHIAMATA fanno riferimento ai record di CLIENTE. Pertanto è necessario trovare un modo per assegnare i valori OID ai record di CLIENTE e per generare riferimenti in CLIENTE\_CHIAMATA.

### Come creare valori OID

Innanzitutto, si utilizzi una vista oggetto per assegnare i valori OID ai record di CLIENTE. Si ricordi che i valori OID vengono assegnati ai record in una tabella oggetto, e che una tabella oggetto, a sua volta, si basa su un tipo di dati astratto. Quindi, per prima cosa occorre creare un tipo di dati astratto che abbia la stessa struttura della tabella CLIENTE:

```
create or replace type CLIENTE_TY as object
(Cliente_ID NUMBER,
 Nome      VARCHAR2(25),
 Via       VARCHAR2(50),
 Citta     VARCHAR2(25),
 Prov      CHAR(2),
 Cap       NUMBER);
```

Ora si crei una vista oggetto basata sul tipo CLIENTE\_TY, mentre si assegnano i valori OID ai record di CLIENTE:

```
create or replace view CLIENTE_OV of CLIENTE_TY
with object identifier (Cliente_ID) as
select Cliente_ID, Nome, Via, Citta, Prov, Cap
from CLIENTE;
```

La prima parte di questo comando `create view` assegna alla vista il nome (`CLIENTE_OV`) e indica a Oracle che la struttura della vista è basata sul tipo di dati `CLIENTE_TY`:

```
create or replace view CLIENTE_OV of CLIENTE_TY
```

La parte successiva del comando `create view` indica al database come dovranno essere costruiti i valori OID per le righe di `CLIENTE`. La clausola `with object identifier` è seguita dalla colonna da utilizzare per l'OID, in questo caso il valore `Cliente_ID`. Questo consente di indirizzare le righe all'interno della tabella `CLIENTE` come se fossero oggetti riga cui è possibile fare riferimento all'interno di una tabella oggetto.

```
with object identifier (Cliente_ID) as
```

**NOTA** *La clausola with object identifier sostituisce la clausola with object OID impiegata nelle versioni precedenti di Oracle. La sintassi di with object OID è ancora supportata per motivi di compatibilità con le versioni precedenti.*

La parte finale del comando `create view` fornisce la query su cui si baserà l'accesso ai dati della vista. Le colonne della query devono coincidere con le colonne nel tipo di dati base della vista:

```
select Cliente_ID, Nome, Via, Citta, Prov, Cap
from CLIENTE;
```

Le righe di `CLIENTE` sono ora accessibili come oggetti riga tramite la vista `CLIENTE_OV`. I valori OID creati per le righe di `CLIENTE_OV` sono detti *pkOID* poiché si basano sui valori della chiave primaria della tabella `CLIENTE`. Alle tabelle relazionali è possibile accedere come a oggetti riga, se per queste tabelle si creano viste oggetto opportune.

### Come creare i riferimenti

Le righe di `CLIENTE_CHIAMATA` fanno riferimento alle righe contenute in `CLIENTE`. Da una prospettiva relazionale, la relazione è determinata dalla chiave esterna che dalla colonna `CLIENTE_CHIAMATA.Cliente_ID` punta alla colonna `CLIENTE.Cliente_ID`. Ora che la vista oggetto `CLIENTE_OV` è stata creata e che è possibile accedere alle righe in `CLIENTE` tramite i valori OID, è necessario creare valori di riferimento in `CLIENTE_CHIAMATA` che facciano riferimento a `CLIENTE`. Posizionati i `REF`, sarà possibile utilizzare la funzione `DEREF` (descritta precedentemente nel capitolo) per accedere ai dati di `CLIENTE` dall'interno di `CLIENTE_CHIAMATA`.

Il comando `create view` per la vista oggetto di `CLIENTE_CHIAMATA` è mostrato nel listato seguente. Questo comando si serve di una nuova funzione, `MAKE_REF`, che verrà descritta dopo il listato seguente.

```
create view CLIENTE_CHIAMATA_OV as
select MAKE_REF(CLIENTE_OV, Cliente_ID) Cliente_ID,
       Numero_Chiamata,
       Data_Chiamata
  from CLIENTE_CHIAMATA;
```

Fatta eccezione per l'operazione MAKE\_REF, questo comando create view ha un aspetto del tutto normale. L'operazione MAKE\_REF è riportata in questa linea:

```
select MAKE_REF(CLIENTE_OV, Cliente_ID) Cliente_ID,
```

La funzione MAKE\_REF accetta come argomenti il nome della vista oggetto cui si fa riferimento e il nome della colonna (o delle colonne) che formano la chiave esterna nella tabella locale. In questo caso, la colonna Cliente\_ID della tabella CLIENTE\_CHIAMATA fa riferimento alla colonna utilizzata come base per la generazione di OID nella vista oggetto CLIENTE\_OV. Quindi a MAKE\_REF vengono passati due parametri: CLIENTE\_OV e Cliente\_ID. Al risultato dell'operazione MAKE\_REF viene assegnato l'alias di colonna Cliente\_ID. Dato che il comando descritto crea una vista, al risultato dell'operazione si deve assegnare un alias di colonna.

Qual è il compito di MAKE\_REF ? MAKE\_REF crea riferimenti (chiamati *pkREF* in quanto basati su chiavi primarie) dalla vista CLIENTE\_CHIAMATA\_OV alla vista CLIENTE\_OV. Ora è possibile eseguire una query sulle due viste come se CLIENTE\_OV fosse una tabella oggetto e CLIENTE\_CHIAMATA\_OV una tabella che contiene un tipo di dati REF che fa riferimento a CLIENTE\_OV.

### Query sulle viste oggetto

Le query sulle viste oggetto con REF rispecchiano la struttura delle query eseguite sui REF della tabella. La funzione DEREF può essere utilizzata per selezionare i valori dei dati cui si fa riferimento, come descritto in precedenza in questo capitolo. Applicata alle viste oggetto, la query avrà l'aspetto seguente:

```
select DEREF(CCov.Cliente_ID)
  from CLIENTE_CHIAMATA_OV CCov
 where Data_Chiamata = TRUNC(SysDate);

Deref(CCov.CLIENTE_ID)(CLIENTE_ID, NOME, VIA, CITTA, PROV, CAP)
-----
CLIENTE_TY(123, 'SIGMUND', '47 HAFFNER RD', 'LEWISTON', 'NJ', .22222)
```

Con la query è stato individuato in CLIENTE\_CHIAMATA il record per cui il valore di Data\_Chiamata era la data di sistema corrente. Quindi si è preso il valore Cliente\_ID da questo record e ne è stato valutato il riferimento. Questo valore Cliente\_ID, come si è dedotto dalla funzione MAKE\_REF, puntava a un valore pkOID nella vista oggetto CLIENTE\_OV. La vista oggetto CLIENTE\_OV ha restituito il record il cui pkOID corrispondeva al valore di riferimento. La funzione DEREF ha restituito quindi il valore della riga cui si faceva riferimento. In questo modo, la query ha restituito delle righe da CLIENTE anche se l'utente si era limitato a eseguire la query solo su CLIENTE\_CHIAMATA.

Le viste oggetto degli oggetti colonna consentono di utilizzare le tabelle come se queste fossero sia tabelle relazionali sia tabelle relazionali a oggetti. Quando vengono estese agli oggetti riga, le viste oggetto consentono di generare valori OID basati sulle relazioni chiave esterna/chiave primaria stabilite. Le viste oggetto consentono di continuare a utilizzare i vincoli già esistenti e i comandi insert, update, delete e select standard. Inoltre permettono di sfruttare le caratteristiche orientate agli oggetti come i riferimenti a tabelle oggetto. Pertanto, le viste oggetto mettono a disposizione un importante ponte tecnologico per le operazioni di migrazione nell'architettura di un database orientato agli oggetti.

Come descritto in precedenza nel capitolo, Oracle esegue join che risolvono i riferimenti definiti nel database. Quando i dati cui si era fatto riferimento vengono recuperati, questi restituiscono l'intero oggetto riga cui si è fatto riferimento. Per fare riferimento ai dati, è necessario

stabilire dei valori pkOID nella tabella che rappresenta la chiave primaria nella relazione, e utilizzare MAKE\_REF per generare riferimenti nella tabella che rappresenta la chiave esterna nella relazione. Quindi si potranno utilizzare i dati come se questi fossero memorizzati in tabelle oggetto.

### 33.4 PL/SQL a oggetti

I programmi PL/SQL sono in grado di usare i tipi di astratti creati. Mentre le versioni precedenti di PL/SQL potevano sfruttare solo i tipi di dati forniti da Oracle (come DATE, NUMBER e VARCHAR2), ora si possono utilizzare anche tipi di dati definiti dall'utente. Il risultato è una mescolanza di SQL, logica procedurale ed estensioni orientate agli oggetti, una combinazione indicata come *PL/SQL a oggetti*.

Il seguente blocco PL/SQL anonimo utilizza i concetti PL/SQL a oggetti. Il tipo di dati astratto CLIENTE\_TY serve come tipo di dati per la variabile “*Cliente1*” e il relativo valore viene popolato tramite una query sulla vista oggetto CLIENTE\_OV.

```
set serveroutput on
declare
    Cliente1      CLIENTE_TY;
begin
    select VALUE(COV)  into Cliente1
        from CLIENTE_OV COV
       where Cliente_ID = 123;
    DBMS_OUTPUT.PUT_LINE(Cliente1.Nome);
    DBMS_OUTPUT.PUT_LINE(Cliente1.Via);
end;
```

L’output prodotto da questo blocco PL/SQL è il seguente:

```
SIGMUND
47 HAFFNER RD
```

Nella prima parte del blocco PL/SQL viene dichiarata una variabile con il tipo di dati CLIENTE\_TY:

```
declare
    Cliente1      CLIENTE_TY;
```

Il valore della variabile “*Cliente1*” viene selezionato dalla vista oggetto CLIENTE\_OV (creata nel paragrafo precedente del capitolo). La funzione VALUE viene utilizzata per recuperare i dati nella struttura del tipo di dati astratto. Dal momento che i dati verranno selezionati nel formato del tipo di dati astratto, si dovrà utilizzare la vista oggetto CLIENTE\_OV creata sulla tabella CLIENTE.

```
begin
    select VALUE(COV)  into Cliente1
        from CLIENTE_OV COV
       where Cliente_ID = 123;
```

I valori “Cliente1.Nome” e “Cliente1.Via” vengono quindi recuperati dagli attributi della variabile *“Cliente1”* e visualizzati. Non è possibile passare l’intera variabile *“Cliente1”* alla procedura **PUT\_LINE**.

```
DBMS_OUTPUT.PUT_LINE(Cliente1.Nome);
DBMS_OUTPUT.PUT_LINE(Cliente1.Via);
end;
```

Questo esempio è volutamente semplice, eppure riesce a illustrare la potenza di PL/SQL a oggetti. Questo linguaggio può essere utilizzato ovunque si usino tipi di dati astratti. Il linguaggio PL/SQL non è più vincolato ai tipi di dati forniti da Oracle e può rispecchiare in modo più accurato gli oggetti del database. In questo esempio è stata eseguita una query su una vista oggetto per mostrare come le query siano in grado di accedere sia a oggetti colonna sia a oggetti riga. Quindi, si potranno selezionare gli attributi del tipo di dati astratto e manipolarli o visualizzarli. Se sono stati definiti dei metodi per il tipo di dati astratto, anch’essi potranno essere applicati.

Per esempio, si possono chiamare i constructor method del tipo di dati all’interno dei blocchi PL/SQL. Nell’esempio seguente, viene definita una variabile di nome *“NuovoCliente”* con il tipo di dati **CLIENTE\_TY**. Quindi, la variabile *“NuovoCliente”* viene impostata uguale a un gruppo di valori tramite il constructor method di **CLIENTE\_TY**. Il gruppo di valori della variabile *“NuovoCliente”* viene quindi inserito tramite la vista oggetto **CLIENTE\_OV**.

```
declare
    NuovoCliente    CLIENTE_TY;
begin
    NuovoCliente :=

        CLIENTE_TY(345,'NuovoCliente','ViaVal', 'Citta','ST',00000);
    insert into CLIENTE_OV
    values (NuovoCliente);
end;
```

Per visualizzare il risultato dell’inserimento, è necessario eseguire una query su **CLIENTE\_OV**:

```
select Cliente_ID, Nome from CLIENTE_OV;
----- -----
CLIENTE_ID  NOME
----- -----
123  SIGMUND
234  EVELYN
345  NuovoCliente
```

Oltre a chiamare i constructor method, si possono chiamare anche i metodi creati sui tipi di dati astratti. Se si prevede di confrontare i valori delle variabili che utilizzano i tipi di dati astratti, è necessario definire mappe o metodi di ordinamento per i tipi di dati. Questa possibilità consente di estendere ulteriormente PL/SQL a oggetti: si definiscono i tipi di dati e le funzioni al livello del database, e questi potranno essere chiamati dall’interno di qualsiasi programma PL/SQL. PL/SQL a oggetti rappresenta un miglioramento significativo rispetto a PL/SQL tradizionale.

### 33.5 Oggetti nel database

Le funzionalità disponibili in Oracle, quali oggetti colonna, oggetti riga ed estensioni a oggetti per PL/SQL, consentono di implementare oggetti nel database senza sacrificare gli investimenti già profusi in analisi e progettazione. Si può continuare a creare sistemi basati su tecniche di progettazione relazionale e adattare specificamente i sistemi creati ai metodi di accesso relazionale. Gli strumenti messi a disposizione da Oracle consentono di creare un livello orientato agli oggetti al di sopra delle tabelle relazionali. Una volta messo in opera questo livello, si potrà accedere ai dati relazionali come se questi fossero memorizzati in un database completamente orientato agli oggetti.

La disponibilità di un livello orientato agli oggetti consente di monetizzare alcuni dei vantaggi di un sistema orientato agli oggetti, come l'astrazione e l'incapsulamento. È possibile applicare i metodi per ciascun tipo di dati astratto attraverso una serie di oggetti implementati in modo coerente e avvalersi del riutilizzo degli oggetti e dell'imposizione degli standard. Nello stesso tempo, si possono sfruttare le caratteristiche relazionali di Oracle. La possibilità di utilizzare all'interno di un'applicazione sia la tecnologia relazionale sia la tecnologia orientata agli oggetti consente di usare lo strumento adatto per l'operazione adatta all'interno del database.

Quando si implementa la sezione a oggetti di un database relazionale a oggetti, si inizia con la definizione dei tipi di dati astratti che costituiscono il nucleo dell'attività. Ogni caratteristica relazionale a oggetti, riferita a oggetti colonna o a oggetti riga, si basa su un tipo di dati astratto. Migliore è la definizione dei tipi di dati e dei relativi metodi, migliori saranno le possibilità di implementare oggetti. Se necessario, è possibile annidare gli oggetti in modo da disporre di più varianti dello stesso tipo di dati "principale". Il risultato sarà un database progettato in modo corretto e in grado di sfruttare le caratteristiche orientate agli oggetti e relazionali messe a disposizione da Oracle nella versione attuale e in quelle future.

Parte quinta

## **JAVA IN ORACLE**



## Capitolo 34

# Un'introduzione a Java

- 34.1 **Confronto tra Java e PL/SQL: introduzione**
- 34.2 **Introduzione**
- 34.3 **Dichiarazioni**
- 34.4 **Comandi eseguibili**
- 34.5 **Classi**

Questo capitolo analizza il modo in cui Java si relaziona alle applicazioni di database di Oracle. Oltre alle applicazioni Oracle, esistono numerosi impieghi di Java, così come esistono molte caratteristiche di questo linguaggio che la maggior parte dei programmatori Oracle ignorerà. Lo scopo di questo capitolo è fornire agli sviluppatori che hanno già una certa familiarità con SQL e PL/SQL una visione delle strutture del linguaggio Java. Questo libro non esamina in dettaglio Java (esistono molti volumi concepiti appositamente per questo scopo); cerca invece di essere un breve riassunto dei componenti del linguaggio Java che gli sviluppatori Oracle utilizzano con maggior frequenza.

Anche se in molti casi PL/SQL e Java sono in correlazione l'uno con l'altro, esistono differenze significative per quanto concerne la terminologia e l'impiego. Questo non dovrebbe rappresentare una sorpresa, poiché questi due linguaggi di programmazione sono stati sviluppati separatamente. Inoltre le capacità a oggetti di PL/SQL non erano previste nella sua implementazione iniziale, mentre Java è nato come linguaggio a oggetti. Per usare Java in modo efficiente, è necessario adottare un approccio differente da quello che potrebbe essere stato impiegato precedentemente con PL/SQL.

Nel Capitolo 27 sono stati introdotti il controllo del flusso e le strutture del linguaggio PL/SQL. Questo capitolo rispecchia quell'approccio: verranno introdotte le strutture impiegate da Java e i metodi base di controllo del flusso messi a disposizione. Quando applicabili, si introduciranno anche le strutture PL/SQL corrispondenti. La lettura di questo capitolo presuppone una conoscenza di PL/SQL (Capitolo 27), package, funzioni e procedure (Capitolo 29), tipi di dati astratti e metodi (Capitolo 30).

### 34.1 Confronto tra Java e PL/SQL: introduzione

Nel confronto tra PL/SQL e Java non si andrà oltre la struttura PL/SQL base, il blocco, prima di incontrare una differenza terminologica significativa tra i due linguaggi. In PL/SQL un *blocco* è una parte strutturata di codice caratterizzata dalle sezioni delle dichiarazioni, dei comandi eseguibili e di gestione delle eccezioni. La struttura di un blocco PL/SQL potrebbe essere rappresentata in questo modo:

```
declare
  <sezione dichiarazioni>
begin
```

```
<comandi eseguibili>
exception
  <gestione delle eccezioni>
end;
```

In Java, il termine *blocco* indica un subset di codice molto più piccolo. In Java, un blocco è una collezione di istruzioni racchiuse tra parentesi graffe, { e }. Per esempio, lo pseudocodice che segue contiene due blocchi Java:

```
if (alcune condizioni) {
  blocco di codice da elaborare se la condizione viene soddisfatta
}
else {
  blocco di codice da elaborare se la condizione non viene soddisfatta
}
```

In PL/SQL, quest'intera sezione di codice sarebbe semplicemente una parte di un blocco più ampio; in Java invece contiene due blocchi separati. In questo capitolo, tutti i riferimenti a “blocchi” indicheranno blocchi Java, a meno che non venga specificato diversamente.

Un'altra differenza importante tra PL/SQL e Java si trova nella dichiarazione delle variabili. In un blocco PL/SQL, le variabili vanno definite prima dell'inizio della sezione dei comandi eseguibili. In un programma Java, le variabili possono essere definite nei punti in cui il programma stesso le richiede. Un programma Java non possiede una sezione di dichiarazioni paragonabile a quella utilizzata nei blocchi PL/SQL.

Nel corso di questo capitolo, si noteranno molte altre differenze tra PL/SQL e Java. L'importante è ricordare che Java non sostituisce PL/SQL all'interno delle applicazioni e che è possibile continuare a utilizzare PL/SQL per le applicazioni Oracle. Per quanto riguarda le operazioni di recupero dati, sarebbe meglio verificare le prestazioni di PL/SQL e Java prima di decidere quale soluzione tecnica adottare.

## 34.2 Introduzione

Per usare gli esempi presentati in questo paragrafo, è necessario possedere una copia del Java Development Kit (JDK), che può essere scaricato gratuitamente dal sito <http://java.sun.com>. Si dovrà installare il JDK sul server su cui verranno eseguiti i programmi Java. Al momento della redazione di questo libro la release di produzione più recente era JDK 1.4; moltissimi sviluppatori però utilizzano ancora JDK 1.3.1. Come minimo, si dovrebbe utilizzare JDK 1.2.2.

## 34.3 Dichiarazioni

In Java è possibile dichiarare le variabili in base alle esigenze. Una variabile ha un nome e un tipo di dato, e serve per memorizzare il valore di un dato durante l'esecuzione del programma. Una variabile ha anche un ambito, che le consente di avere un accesso pubblico o privato, analogamente alle procedure che possono avere nei package delle variabili private.

Ecco alcuni esempi di dichiarazioni di variabili Java:

```
char    aCharVariable = 'J';
boolean aBooleanVariable = false;
```

Per convenzione, le variabili Java incominciano sempre con una lettera minuscola, come mostrato in questo esempio. Nell'esempio i tipi di dati sono CHAR e BOOLEAN, e ad ogni variabile viene assegnato un valore iniziale. Si osservi che in Java il carattere di assegnamento è `=`, mentre in PL/SQL può essere `:= o =`. Ogni dichiarazione termina con un segno di due punti.

Nella Tabella 34.1 sono elencati i tipi di dati primitivi disponibili in Java.

Oltre ai tipi di dati primitivi elencati nella Tabella 34.1, è possibile utilizzare i tipi di dati riferimento basati sul contenuto delle variabili. Si potrebbe osservare che per le stringhe di caratteri a lunghezza variabile non esistono tipi di dati primitivi: infatti Java mette a disposizione una classe String proprio per questo scopo. Le classi verranno analizzate più avanti in questo capitolo.

## 34.4 Comandi eseguibili

L'assegnamento di valori alle variabili avviene attraverso l'uso di espressioni e istruzioni. Tra gli operatori aritmetici supportati da Java troviamo i seguenti:

OPERATORE	DESCRIZIONE
*	Moltiplicazione
/	Divisione
+	Addizione
-	Sottrazione
%	Modulo

Per semplificare la codifica, oltre a questi operatori è possibile utilizzare gli operatori unari di Java. Per esempio, tramite l'operatore `++` è possibile incrementare il valore delle variabili:

```
aLoopCounter = aLoopCounter++;
```

**Tabella 34.1** Tipi di dati Java primitivi.

TIPO DI DATO	DESCRIZIONE
byte	Intero della lunghezza di un byte.
short	Intero short.
int	Intero.
long	Intero long.
float	Numero reale in virgola mobile a singola precisione.
double	Numero reale in virgola mobile a doppia precisione.
char	Un singolo carattere Unicode.
boolean	Un valore booleano, vero o falso.

oppure, si potrebbe scrivere semplicemente:

```
aLoopCounter++
```

**NOTA** *L'operatore ++ prende il nome di operatore unario perché ha un solo operando. Se un operatore richiede due operandi (come \*), prende il nome di operatore binario.*

In PL/SQL, tutto questo sarebbe stato scritto come:

```
aLoopCounter := aLoopCounter +1;
```

Se l'operatore **++** viene inserito prima del nome della variabile, allora si tratta di un operatore di *incremento prefisso* invece che di un operatore di *incremento suffisso*:

```
int anotherCounter = ++aLoopCounter;
```

In questo esempio, la variabile *anotherCounter* viene impostata al valore incrementato *aLoopCounter*. Per contrasto:

```
int anotherCounter = aLoopCounter++;
```

imposta *anotherCounter* al valore *aLoopCounter*, prima di essere incrementato. Tramite l'operatore unario **--**, è possibile decrementare il valore di una variabile:

```
aLoopCounter = aLoopCounter--;
```

Inoltre, è possibile combinare l'assegnamento dei valori con un'operazione. Per esempio, invece di scrivere:

```
aNumberVariable = aNumberVariable * 2;
```

si può scrivere:

```
aNumberVariable *= 2;
```

**/=**, **%=**, **+=** e **-=** rendono possibili combinazioni simili.

Se si eseguono più operazioni aritmetiche, si dovrebbero inserire delle parentesi per indicare chiaramente l'ordine in cui le operazioni dovrebbero essere valutate, come nell'esempio seguente:

```
aNumberVariable = (aNumberVariable*2) +2;
```

L'aggiunta di un punto e virgola al termine di queste operazioni le trasforma in *istruzioni*, unità complete di esecuzione.

## Logica condizionale

Con le espressioni e le istruzioni è possibile valutare le variabili. A tal fine si potrebbero usare una o più delle classi fondamentali che fanno parte di Java. Un elenco completo di questi metodi va ben oltre lo scopo di questo libro; si faccia riferimento al sito di documentazione su Java di Sun, oppure a uno dei numerosi manuali su Java che trattano l'argomento.

**NOTA** *Il numero di classi fornite cambia enormemente con ogni release di Java.*

Nel listato seguente, il metodo Character.isUpperCase viene utilizzato per valutare *aCharVariable*, dichiarata precedentemente:

```
if (Character.isUpperCase(aCharVariable)) {
    alcuni comandi da eseguire
}
```

Se il valore *aCharVariable* è scritto in maiuscolo, i comandi contenuti nel blocco seguente verranno eseguiti; in caso contrario, il flusso proseguirà nella parte successiva del programma.

Ecco la sintassi generale per le clausole if:

```
if (espressione) {
    istruzione
}
```

Si noti l'assenza di una clausola *then*. Se l'espressione valutata è vera, il blocco seguente verrà eseguito automaticamente.

È anche possibile che delle clausole *else* valutino condizioni differenti, come mostrato nel listato seguente:

```
if (espressione) {
    istruzione
}
else {
    istruzione
}
```

Se si devono verificare più condizioni, è possibile utilizzare la clausola *else if* per valutarle una alla volta, terminando con una clausola *else*. Il listato seguente illustra come usare le clausole *else if*.

```
if (aCharVariable == 'V') {
    istruzione
}
else if (aCharVariable == 'J') {
    istruzione
}
else {
    istruzione
}
```

Oltre a supportare le clausole if per la logica condizionale, Java presenta una clausola switch. Utilizzata insieme alla sua clausola break e alle etichette delle istruzioni, la clausola switch è in grado di avvicinarsi alla funzionalità delle clausole goto (che Java non supporta). Per prima cosa, si analizzi un semplice esempio di uso di switch. Con l'istruzione case, è possibile valutare più valori della variabile *aMonth*. In base al valore di *aMonth*, viene visualizzato il nome del mese, e l'elaborazione lascia il blocco switch tramite break.

```
int aMonth=2;
switch (aMonth) {
    case 1:
        System.out.println("Gennaio");
        break;
    case 2:
        System.out.println("Febbraio");
```

```

        break;
    case 3:
        System.out.println("Marzo");
        break;
    default:
        System.out.println("Fuori dall'intervallo");
        break;
}

```

Questo tipo di codice è molto più complesso da scrivere rispetto a un semplice TO\_CHAR, tuttavia mostra l'impiego di switch. L'operatore switch accetta la variabile *aMonth* come proprio input, e ne valuta il valore. Se questo valore corrisponde a un valore di case, allora verrà chiamato il metodo per visualizzare il nome del mese, mentre il controllo dell'elaborazione lascerà il blocco switch tramite break. Se questo valore non corrisponde a una delle clausole case, allora verrà eseguita l'opzione default.

Dove va a finire il controllo? Per default, passa alla sezione successiva del programma. Tuttavia esiste la possibilità di creare etichette per le sezioni di codice e passare il nome di queste etichette alla clausola break. Il listato seguente mostra un esempio di un'etichetta e di una clausola break:

```

somelabel:
if (aCharVariable == 'V') {
    istruzione
}
else {
    istruzione
}
int aMonth=2;
switch (aMonth) {
    case 1:
        System.out.println("Gennaio");
        break somelabel;
    case 2:
        System.out.println("Febbraio");
        break someotherlabel;
    case 3:
        System.out.println("Marzo");
        break somethirdlabel;
    default:
        System.out.println("Fuori dall'intervallo");
        break;
}

```

In questo esempio, viene valutata prima la clausola if e poi la clausola switch. Si noti che la variabile *aMonth* viene dichiarata solo quando è necessaria. Per questo esempio, alla variabile *aMonth* viene assegnato un valore pari a 2: in questo modo il programma visualizzerà la parola "Febbraio" e passerà poi alla sezione di codice identificata dall'etichetta *someotherlabel*. Se il valore di *aMonth* fosse stato 1, la clausola if posizionata all'inizio del listato di codice sarebbe stata eseguita nuovamente.

Java supporta anche un operatore *ternario*, ossia un operatore che accetta tre operandi e la cui funzione è simile a quella di DECODE. L'operatore ternario funziona come una combinazione if-else in linea e valuta un'espressione. Se l'espressione è vera, allora verrà restituito il secondo operando, altrimenti verrà restituito il terzo. La sintassi è simile alla seguente:

*espressione* ? *operando1* : *operando2*

Il listato seguente mostra un esempio di operazione ternaria:

```
aStatusVariable = (aCharVariable == 'V') ? "OK" : "Non valido";
```

In questo esempio, viene valutata l'espressione:

```
(aCharVariable == 'V')
```

Se è vera, viene restituito “OK”; in caso contrario, viene restituito “Non valido”.

L'operatore ternario può essere utilizzato per simulare l'uso di una funzione GREATEST:

```
double greatest = (a > b) ? a : b;
```

In questo esempio, viene valutata l'espressione ( $a > b$ ), in cui  $a$  e  $b$  sono i nomi delle variabili. Se quest'espressione è vera, verrà restituito il valore di  $a$ ; in caso contrario verrà restituito il valore di  $b$ .

**NOTA** Java ha un metodo “greatest” per i valori numerici, ossia il metodo Max contenuto nella classe `java.lang.Math`. L'esempio ternario riportato nel listato precedente può essere riscritto `double greatest = Math.max(a, b);`

Con le operazioni AND e OR è possibile combinare più controlli logici. In Java, un'operazione AND è rappresentata dall'operatore `&&`:

```
if (aCharVariable == 'V' && aMonth == 3) {
    istruzione
}
```

L'operazione OR è rappresentata da `||`, come mostrato nel listato seguente:

```
if (aCharVariable == 'V' || aMonth == 3) {
    istruzione
}
```

Per motivi legati alle prestazioni, le operazioni `&&` e `||` vengono eseguite solo se necessarie; la seconda espressione non viene valutata se la prima stabilisce già il risultato. Per quanto riguarda l'operatore AND, se il primo valore è falso, la valutazione terminerà in quel punto e il valore booleano falso verrà restituito. Per quanto riguarda l'operatore OR, se la prima espressione è falsa, la seconda verrà comunque valutata perché è sufficiente che una sola delle condizioni di verifica sia vera per consentire la restituzione del valore booleano vero. Se il primo valore dell'operatore OR fosse vero, non sarebbe necessario valutare le espressioni successive e il valore vero verrebbe restituito.

Gli operatori `&` e `|` prendono il nome di AND e OR bit per bit: non si tratta di operatori a corto circuito come `&&` e `||`. Verranno valutate sempre tutte le espressioni. Questa funzionalità può essere particolarmente utile quando si verificano i valori di restituzione delle chiamate ai metodi ed è necessario assicurarsi che i metodi siano stati eseguiti.

## Cicli

Java supporta tre tipi principali di cicli: i cicli WHILE, DO-WHILE e FOR. In questo paragrafo, verranno presentati esempi per ogni tipo di ciclo. Dato che Java non è stato concepito come estensione di SQL, esso non supporta i cicli FOR a cursore, come avviene in PL/SQL.

## I cicli WHILE e DO-WHILE

Una clausola while valuta una condizione; se la condizione è vera, verrà eseguito il blocco associato di istruzioni. La sintassi di un ciclo WHILE presenta la forma seguente:

```
while (espressione) {
    istruzione
    istruzione
}
```

Il ciclo WHILE esegue ripetutamente il blocco di istruzioni (la sezione racchiusa tra le parentesi graffe { and }) fino a quando *espressione* non è più vera:

```
int aNumberVariable = 3;
while (aNumberVariable < 7) {
    qualche_istruzione di_elaborazione
    aNumberVariable++;
}
```

In questo esempio, viene creata e inizializzata una variabile contatore (*aNumberVariable*). Successivamente, il valore della variabile viene valutato tramite la clausola while. Se questo valore è minore di 7, allora il blocco di istruzioni associato verrà eseguito. Poiché fa parte di quel blocco, la variabile viene incrementata. Quando il blocco termina, la variabile viene valutata nuovamente e il ciclo continua.

**NOTA** Per esempi sull'elaborazione del ciclo WHILE, fare riferimento al paragrafo “Classi” più avanti in questo capitolo.

Questo esempio potrebbe anche essere scritto sotto forma di ciclo DO-WHILE:

```
int aNumberVariable = 3;
do {
    qualche_istruzione di_elaborazione
    aNumberVariable++;
} while (aNumberVariable < 7);
```

In un ciclo DO-WHILE, il valore dell'espressione non viene valutato fino a quando il blocco non è stato elaborato almeno una volta.

## I cicli FOR

Un ciclo FOR può essere utilizzato per eseguire ripetutamente un blocco di codice. In un ciclo FOR, si specificano un valore iniziale, un valore finale e un incremento per il contatore del ciclo. Il contatore del ciclo verrà incrementato del valore specificato ogni volta che si esegue il ciclo fino a quando non si raggiunge il valore finale. Ecco la sintassi di un ciclo FOR:

```
for (inizializzazione; terminazione; incremento) {
    istruzione;
}
```

L'esempio di WHILE riportato nel paragrafo precedente può essere riscritto con una clausola for, come mostrato nel listato seguente:

```
for (int aNumberVariable=3; aNumberVariable<7; aNumberVariable++) {
    qualche_istruzione di_elaborazione
}
```

La versione della clausola `for` impiegata per questo esempio è molto più breve della versione `while`. In questo esempio, la variabile `aNumberVariable` viene dichiarata e inizializzata all'interno del ciclo `FOR`. Il blocco verrà eseguito fino a quando il valore della variabile non raggiunge 7. Per ogni passaggio nel ciclo, il valore della variabile viene aumentato di 1 attraverso l'operatore unario `++`.

All'interno di un ciclo, è possibile ricorrere alla clausola `continue` per passare a un'altra istruzione contenuta nel ciclo. Se si utilizza la clausola `continue` da sola, il flusso del processo salterà alla fine del corpo del ciclo e valuterà il test finale del ciclo. Se invece si utilizza `continue` insieme a un'etichetta (come mostrato in precedenza nel paragrafo relativo alla clausola `switch`) l'elaborazione riprenderà dall'iterazione successiva del ciclo con etichetta.

## Cursori e cicli

In PL/SQL i cicli FOR a cursore possono servire per iterare attraverso i risultati di una query. I cicli WHILE e FOR di Java possono essere utilizzati per simulare la funzionalità di un ciclo FOR a cursore PL/SQL. Gli esempi forniti con Oracle9i mostrano come raggiungere questo risultato. Gli esempi presentati in Java si trovano nella directory `/jdbc/demo` nella directory home del software Oracle. Il file `Employee.java` situato in questa directory crea la classe `Employee` in Java. Come parte di quella classe, l'esempio esegue i comandi seguenti:

```
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery ("select ENAME from EMP");
//itera attraverso l'insieme di risultati e visualizza i nomi degli impiegati
while (rset.next ())
    System.out.println (rset.getString (1));
```

Questo esempio presuppone che una connessione al database sia già stata stabilita. La prima linea dell'esempio crea una variabile di nome `stmt`, che si basa su una variabile di connessione chiamata `conn` (che non è stata riportata in questo listato parziale). La seconda linea dell'esempio crea una variabile di nome `rset`, basata sul gruppo di risultati che verrà restituito dalla query su `stmt`. Dopo un commento (che ha come prefisso `//`), una clausola `while` recupera ogni linea nel gruppo dei risultati. Se l'operazione `FETCH` recupera una linea dal gruppo dei risultati, verrà visualizzato `ENAME` da quella linea `EMP`; altrimenti il ciclo terminerà.

Dal punto di vista della programmazione, è necessario essere specifici per quanto concerne i dati recuperati. L'esempio riportato da Oracle può essere modificato per utilizzare il nome della colonna restituita, invece del numero assoluto di una colonna, come mostrato nel listato seguente:

```
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery ("select ENAME from EMP");
// iterat attraverso l'insieme di risultati e visualizza i nomi degli impiegati
while (rset.next ()) {
    System.out.println (rset.getString ("ENAME"));
}
```

I dimostrativi di JDBC (Java Database Connectivity, fare riferimento al Capitolo 35) messi a disposizione da Oracle mostrano alcuni esempi di query che richiedono chiamate di procedure, LOB (Large Object) e il linguaggio DML (Data Manipulation Language). Nella maggior parte dei casi, il codice Java si comporterà in modo simile agli esempi precedenti: si scriva il codice per il controllo del flusso del processo in Java e gli si passi un'istruzione da eseguire. In base al risultato di quest'istruzione, grazie alle clausole `while`, `for`, `if`, `break`, `continue` e `switch`, come mostrato in precedenza, è possibile decidere l'esecuzione di azioni differenti.

## Gestione delle eccezioni

Java mette a disposizione un efficace insieme di routine per la gestione delle eccezioni e permette all'utente di creare procedure complesse per la gestione degli errori. In PL/SQL si "solleva" (*raise*) un'eccezione, mentre in Java si "lanciano" (*throw*) delle eccezioni. Se si lancia un'eccezione, è necessario ricorrere a una clausola *catch* Java per catturare ed elaborare correttamente questa eccezione.

La sintassi per la gestione delle eccezioni di Java si basa su tre blocchi: *try*, *catch* e *finally*. Il blocco *try* include le istruzioni che potrebbero lanciare un'eccezione. Il blocco *catch*, che segue immediatamente il blocco *try*, associa i gestori di eccezioni alle eccezioni che potrebbero essere lanciate dal blocco *try*. Infine, il blocco *finally* pulisce qualsiasi risorsa di sistema che non sia stata pulita correttamente durante l'elaborazione del blocco *catch*. La struttura generale è la seguente:

```
try {
    istruzioni
}
catch ( ) {
    istruzioni
}
catch ( ) {
    istruzioni
}
finally {
    istruzioni
}
```

Per esempio, il codice seguente deriva dal file di esempio PLSQL.java fornito come parte integrante di Oracle9i:

```
try {
    Statement stmt = conn.createStatement ();
    stmt.execute ("drop table plsqltest");
}
catch (SQLException e) { }
```

Qui viene creata una variabile di nome *stmt* per eseguire un comando *drop table*. Se l'esecuzione fallisce, per esempio nel caso in cui la tabella non esista, come verrà elaborato questo errore? La clausola *catch* comunica a Java di gestirlo tramite un oggetto di eccezione chiamato *SQLException*, un componente standard dell'implementazione SQL di Java. Come mostrato nell'esempio, la clausola *catch* accetta due parametri (un oggetto di eccezione e il nome di una variabile) ed è facoltativamente seguita da un blocco di istruzioni da eseguire.

Il blocco *try* può contenere più istruzioni, oppure si possono creare blocchi *try* separati per ogni istruzione. Generalmente è più facile controllare la gestione delle eccezioni se si consolida no i blocchi *catch*, pertanto il consolidamento dei blocchi *try* aiuterà a gestire il codice mentre cambia e cresce in dimensioni.

Il blocco *finally* ripulisce lo stato della sezione di codice corrente prima di passare il controllo a qualsiasi parte successiva del programma. In fase di esecuzione, il contenuto del blocco *finally* viene eseguito sempre, a prescindere dal risultato del blocco *try*. Per esempio, si potrebbe usare il blocco *finally* per chiudere una connessione al database.

## Parole riservate

Le parole riservate contenute in Java sono elencate nella Tabella 34.2. Non è possibile utilizzare queste parole come nomi di classi, metodi o variabili.

### 34.5 Classi

Nei paragrafi precedenti di questo capitolo sono state introdotte le strutture sintattiche di base per Java. In questo paragrafo si imparerà a utilizzare questa sintassi per creare e usare gli oggetti. Nel listato seguente viene proposto un programma semplice utilizzato per visualizzare la parola “Oracle”:

```
public class HelloOracle {
    public static void main (String[] args) {
        System.out.println("Oracle");
    }
}
```

Questo esempio crea una *classe* chiamata HelloOracle, all'interno della quale viene creato un metodo pubblico di nome *main*. Il metodo main visualizza la parola “Oracle”. Ciò è molto più complesso di:

```
select 'Oracle' from dual;
```

**Tabella 34.2** Parole riservate in Java.

abstract	else	interface	super
boolean	extends	long	switch
break	false	native	synchronized
byte	final	new	this
case	finally	null	throw
catch	float	package	throws
char	for	private	transient
class	goto	protected	true
const	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while
double	int	strictfp	

tuttavia la versione Java possiede delle soluzioni che invece mancano nella versione SQL. HelloOracle è una classe e, in quanto tale, i suoi metodi possono essere chiamati dall'interno di altre classi. HelloOracle potrebbe poi avere più metodi, anche se in questo esempio possiede solamente il metodo main.

La dichiarazione del metodo main contiene un certo numero di parole chiave (public static void main) da imparare:

public	La definizione della classe come public consente a tutte le altre classi di chiamare questo metodo.
static	Dichiara che si tratta di un metodo di classe; static serve per dichiarare i metodi di classe. Un'opzione aggiuntiva è la parola chiave final per quei metodi e variabili i cui valori sono immutabili (analogia all'opzione constant di PL/SQL).
void	Specifica il tipo di dato del valore di restituzione dalla procedura. Dato che questa procedura non restituisce valori, il suo tipo è void.

In questo esempio, si attribuisce il nome main al metodo perché questo semplificherà l'esecuzione del metodo della classe. Una volta creata la classe, la si potrà compilare e caricare.

**NOTA** *Per compilare ed eseguire una classe Java, è necessario possedere sul proprio server il JDK. Gli esempi seguenti partono dal presupposto che i file binari che fanno parte di quel kit si trovino in una directory che verrà cercata automaticamente tramite la variabile ambientale PATH, e che la directory contenente i file .class sia inclusa nella variabile ambientale CLASSPATH. I programmi cui si fa riferimento in questa sede, (javac e java) si trovano nella sottodirectory /bin della directory del software JDK.*

Per prima cosa, si salvi il programma sotto forma di file di solo testo, di nome HelloOracle.java. Quindi, lo si compili:

```
javac HelloOracle.java
```

Sul server, verrà creato un nuovo file, chiamato HelloOracle.class. Quindi, si potrà eseguire la classe:

```
java HelloOracle
```

L'output verrà così visualizzato:

```
Oracle
```

Come mostrato nell'esempio di HelloOracle, per prima cosa alla classe viene assegnato un nome, insieme ad altri attributi facoltativi che riguardano la sua ereditarietà di attributi di altre classi. Nel corpo della classe vengono definiti tutti i metodi e le variabili della classe stessa. Come avviene per i tipi di dati astratti in Oracle, anche alle classi Java sono associati dei metodi che manipolano i dati a esse correlati.

Si analizzi un esempio più complesso. Il listato seguente è un programma Java che calcola l'area di un cerchio, avendo come input il valore del raggio (per la versione PL/SQL, si faccia riferimento al Capitolo 27):

```
// AreaOfCircle.java
//
public class AreaOfCircle {
```

```

public static void main(String[] args) {
    try {
        int input = Integer.parseInt(args[0]);
        double result=area(input);
        System.out.println(result);
    }
    catch (ArrayIndexOutOfBoundsException oob) {
        System.out.println("Non è stato fornito il raggio del cerchio.");
        System.out.println("Utilizzo: java AreaOfCircle 10");
    }
    catch (NumberFormatException nfe) {
        System.out.println("Inserire un numero valido per il raggio.");
        System.out.println("Utilizzo: java AreaOfCircle 10");
    }
}
} // main

public static double area (int rad) {
    double pi=3.1415927;
    double areacircle=pi*rad*rad;
    return areacircle;
} // area
} //AreaOfCircle

```

**NOTA** Si controlli di aver inserito una parentesi graffa di chiusura, }, per ogni blocco. Negli esempi proposti in questo capitolo, le parentesi di chiusura si trovano sempre in una linea a parte per semplificare il debug e l'elaborazione degli eventuali errori relativi al controllo del flusso.

Innanzitutto si osservi che la classe e il nome del file devono corrispondere. In questo caso, la classe prende il nome di `AreaOfCircle`, pertanto questo testo verrà memorizzato in un file di nome `AreaOfCircle.java`. La classe ha due metodi, chiamati `main` e `area`. Quando si esegue la classe, il metodo `main` viene eseguito automaticamente. Il metodo `main` accetta la stringa fornita come `input` e la analizza come un intero nella variabile `input`:

```
int input=Integer.parseInt(args[0]);
```

Esistono almeno due eccezioni possibili che potrebbero essere lanciate durante l'esecuzione di questo codice. La prima eccezione gestita, `ArrayIndexOutOfBoundsException`, si verifica nel momento in cui si cerca di far riferimento a una posizione in un array che non esiste. Dato che il codice fa esplicitamente riferimento alla prima posizione dell'array, chiamata `args` (`args[0]`), se non si passano dati al programma, verrà lanciata un'eccezione. L'altro possibile errore, `NumberFormatException`, si verificherebbe se si passasse un valore non numerico. Per esempio, se si eseguisse il programma inserendo "java `AreaOfCircle XYZ`" al posto di "java `AreaOfCircle 123`", verrebbe lanciata l'eccezione `NumberFormatException`, in quanto `XYZ` non è un numero valido. Entrambe le eccezioni sono gestite dalle due istruzioni `catch` che forniscono all'utente delle informazioni utili relative alla causa dell'errore:

```

catch (ArrayIndexOutOfBoundsException oob) {
    System.out.println("Non è stato fornito il raggio del cerchio.");
    System.out.println("Utilizzo: java AreaOfCircle 10");
}
catch (NumberFormatException nfe) {
    System.out.println("Non è stato inserito un numero valido per il raggio.");
    System.out.println("Utilizzo: java AreaOfCircle 10");
}

```

Quindi una variabile di nome *result* viene dichiarata e impostata al valore restituito dalla chiamata ad *area(input)*:

```
double result=area(input);
```

Per elaborare questa direttiva, viene eseguito il metodo *area*, utilizzando il valore della variabile *input* come *input*. Ecco il metodo *area*:

```
public static double area (int rad) {
    double pi=3.1415927;
    double areacircle=pi*rad*rad;
    return areacircle;
}
```

Nella sua definizione, il metodo *area* specifica che accetterà una variabile, un tipo intero di nome *rad*. Questo valore della variabile (passato dalla variabile *input* nel metodo *main*) verrà poi elaborato e una variabile di nome *areacircle* verrà calcolata e restituita a *main*. In *main*, verrà visualizzato questo valore:

```
double result=area(input);
System.out.println(result);
```

Verificare il programma è semplice:

```
javac AreaOfCircle.java
java AreaOfCircle 10
```

Ecco l'output:

```
314.15927
```

Si provi ad ampliare questo esempio in modo che includa anche l'elaborazione di un ciclo WHILE. In questo esempio, il metodo *area* verrà chiamato ripetutamente fino a quando il valore di *input* risultante non supererà un determinato valore:

```
// AreaOfCircleWhile.java
//
public class AreaOfCircleWhile {
    public static void main(String[] args) {
        try {
            int input=Integer.parseInt(args[0]);
            while (input < 7) {
                double result=area(input);
                System.out.println(result);
                input++;
            }
        }
        catch (ArrayIndexOutOfBoundsException oob) {
            System.out.println("Non è stato fornito il raggio del cerchio.");
            System.out.println("Utilizzo: java AreaOfCircle 10");
        }
        catch (NumberFormatException nfe) {
            System.out.println("Inserire un numero valido per il raggio.");
            System.out.println("Utilizzo: java AreaOfCircle 10");
        }
    }
}
```

---

```

} /// main
public static double area (int rad) {
    double pi=3.1415927;
    double areacircle=pi*rad*rad;
    return areacircle;
} /// area
} /// AreaOfCircleWhile

```

In questo listato, il metodo `area` e i blocchi per la gestione delle eccezioni sono rimasti immutati rispetto all'esempio precedente. La modifica riguarda il metodo `main`:

```

int input=Integer.parseInt(args[0]);
while (input < 7) {
    double result=area(input);
    System.out.println(result);
    input++;
}

```

Finché il valore di `input` è minore di 7, verrà elaborato. Dopo aver calcolato e visualizzato l'area per quel determinato valore del raggio, il valore di `input` viene incrementato:

```
input++;
```

e il ciclo valutato nuovamente. Nel listato seguente viene mostrato dell'output di esempio.

```

javac AreaOfCircleWhile.java
java AreaOfCircleWhile 4

```

```

50.2654832
78.5398175
113.0973372

```

L'output mostra i calcoli dell'area per valori del raggio di `input` pari a 4, 5 e 6.

Questo esempio non è completo come un programma finito: per esempio, potrebbe essere necessaria una maggiore gestione delle eccezioni per gestire i valori non interi; tuttavia è la struttura l'elemento più importante che si deve ricavare da questo esempio. Oltre ai metodi mostrati in questo esempio, si potrebbe generare un costruttore per la classe `AreaOfCircleWhile`. Tutte le classi Java possiedono dei costruttori per inizializzare nuovi oggetti che si basano sulla classe. Il nome del costruttore è lo stesso di quello della classe. Se si crea una classe priva di costruttori, Java creerà un costruttore predefinito al momento dell'esecuzione. Come per i metodi, il corpo del costruttore è in grado di contenere dichiarazioni di variabili locali, cicli e blocchi di istruzioni. Il costruttore inizializza queste variabili per la classe.

Nei prossimi capitoli si imparerà a implementare Java in Oracle all'interno di stored procedure e tramite il SQLJ/JDBC. Questi capitoli richiedono la conoscenza base di Java, fornita in questo capitolo. Per ulteriori informazioni sul linguaggio Java, fare riferimento al sito Web di Sun Microsystems e ai numerosi libri dedicati a Java.



## Capitolo 35

# Programmazione JDBC e SQLJ

- 35.1 **Introduzione**
- 35.2 **Uso delle classi JDBC**
- 35.3 **SQLJ**
- 35.4 **Uso delle classi SQLJ**

**J**ava Database Connectivity (JDBC) e SQLJ si basano su Java e sui principi fondamentali di programmazione descritti in precedenza in questo libro. Le discussioni e gli esempi di questo capitolo suppongono una certa familiarità con la sintassi e le strutture Java descritte nel Capitolo 34. Questo capitolo non analizza tutti gli aspetti di JDBC e SQLJ, concentrandosi principalmente sulla configurazione e l'utilizzo base di questi strumenti.

JDBC può servire per eseguire istruzioni SQL dinamiche in programmi Java. Oracle mette a disposizione dei file di esempio con l'installazione del suo software standard. Il listato seguente è tratto dal file dimostrativo Employee.java, situato assieme al file /jdbc/demo/demo.zip nella directory home del software Oracle. Il file Employee.java esegue i comandi seguenti dopo aver stabilito una connessione al database:

```
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery ("select ENAME from EMP");
//itera attraverso l'insieme dei risultati e visualizza i nomi degli impiegati
while (rset.next ())
    System.out.println (rset.getString (1));
```

Quando si esegue un'istruzione SQL tramite JDBC, in questo caso select Ename from EMP, l'istruzione SQL non viene sottoposta a controllo per verificare eventuali errori fino a quando non viene eseguita. Il programma JDBC, che contiene il comando SQL, verrà compilato anche se l'istruzione SQL non è valida.

Al contrario, SQLJ esegue un passaggio di precompilazione, che prevede il controllo della sintassi del codice SQL incorporato all'interno del programma. SQLJ effettua anche il controllo di tipo e dello schema per garantire che i dati vengano recuperati ed elaborati in modo corretto. Terminata la fase di precompilazione di SQLJ, il risultato sarà un programma Java eseguito attraverso le chiamate JDBC. Anche se distinti, SQLJ e JDBC sono comunque correlati.

## 35.1 Introduzione

Per usare gli esempi presentati in questo paragrafo, è necessario possedere una copia del Java Development Kit (JDK), che può essere scaricato gratuitamente dal sito <http://java.sun.com>. Si dovrà installare il JDK sul server su cui verranno eseguiti i programmi Java. Al momento della redazione di questo libro la release di produzione più recente era JDK 1.4; moltissimi sviluppatori

però utilizzano ancora JDK 1.3.1. Come minimo, si dovrebbe utilizzare JDK 1.2.2. Gli esempi di questo capitolo richiedono l'utilizzo di JDK 1.3.1.

## Passaggi aggiuntivi per gli utenti NT

Se si accede a Oracle9i, si dovrà impostare la variabile ambientale CLASSPATH per trovare il file classes12.zip (per JDK 1.2.2 o 1.3.1) fornito da Oracle. Questo file si trova nella directory /jdbc/lib sotto la directory home di Oracle, oppure lo si può scaricare dal sito <http://technet.oracle.com>. Per JDK 1.1.8, si utilizzi il file classes11.zip, situato nella directory /jdbc/lib.

**NOTA** *Questo passaggio non viene eseguito dall'installazione standard di Oracle9i, quindi lo si dovrà effettuare manualmente. Se le variabili di sistema non sono impostate correttamente, non si potranno compilare le classi Java che utilizzano le classi JDBC di Oracle.*

Per impostare le variabili di sistema, fare clic sull'icona Sistema nel Pannello di controllo, scegliere la scheda Ambiente per elencare le variabili ambientali e le loro definizioni. La variabile PATH dovrebbe essere già impostata, quindi è sufficiente selezionarla e modificarne il valore. Aggiungere la nuova voce alla fine dell'elenco, come mostrato nel listato seguente. Questo valore dovrebbe essere separato dagli altri contenuti nell'elenco con un punto e virgola.

```
E:\Oracle\Ora91\jdbc\lib\classes12.zip
```

Sostituire "E:\Oracle\Ora91" con la directory home del software Oracle. All'impostazione di PATH si dovrebbe aggiungere anche la directory dei file binari JDK. Il listato successivo mostra l'impostazione di PATH ampliata in modo da includere una directory di file binari JDK 1.3.1 (sostituire "E:\jdk131" con la directory in cui è stato installato JDK):

```
E:\jdk131\bin
```

A questo punto occorre creare una variabile ambientale di nome CLASSPATH, se non ne esiste già una (per la configurazione iniziale di Java non ne esisterà nessuna). Questa variabile deve avere due voci, separate da un punto e virgola. La prima voce dev'essere un punto, che denota la directory corrente. La seconda voce dev'essere la directory per il file classes12.zip, che utilizza la stessa forma e il medesimo valore usati per il valore di PATH:

```
.;E:\Oracle\Ora91\jdbc\lib\classes12.zip
```

Se CLASSPATH non è impostata in modo corretto, quando si eseguirà una classe compilata si otterrà un errore NoClassDefFoundError.

**NOTA** *È necessario utilizzare una versione di JDK compatibile con la release di Oracle impiegata. Se si usa una nuova release di JDK con una release più datata dei driver di Oracle, si potrebbe andare incontro a errori di "violazione di accesso" durante l'esecuzione dei programmi.*

## Verifica della connessione

Oracle mette a disposizione un programma di esempio chiamato JdbcCheckup.java che serve per verificare la configurazione di JDBC. Questo file potrebbe essere contenuto in un file ZIP

(demo.zip nella directory /jdbc/demo), in questo caso lo si dovrà estrarre prima di eseguire il programma. Compili ed eseguire la classe JdbcCheckup.java:

```
javac JdbcCheckup.java
java JdbcCheckup
```

Durante l'esecuzione di JdbcCheckup, verranno richiesti un nome utente, una password e una stringa di connessione per un database. Queste informazioni di connessione serviranno per tentare una connessione; in caso positivo, questo tentativo restituirà l'output seguente:

```
Connecting to the database...Connecting...
connected.
Hello World.
Your JDBC installation is correct.
```

Se non si ricevono informazioni relative alla correttezza dell'installazione, si dovrà procedere al controllo della configurazione. Tra i problemi più comuni ci sono le variabili ambientali (PATH e CLASSPATH) impostate in modo non corretto e versioni non corrispondenti dei driver di connessione al database. Se si modificano i valori delle variabili ambientali, perché queste modifiche abbiano effetto si dovranno chiudere e riavviare le finestre dei comandi.

## 35.2 Uso delle classi JDBC

JDBC viene implementato in Oracle attraverso un gruppo di classi fornite da Oracle i cui nomi incominciano con oracle.sql, mentre le classi JDBC standard, fornite con il JDK, iniziano con java.sql. La classe JdbcCheckup.java, mostrata nel listato seguente, offre una guida valida per i programmatori JDBC principianti. La classe JdbcCheckup.java richiede l'uso di Oracle Net per stabilire una connessione al database.

**NOTA** Il codice seguente è fornito da Oracle. Gli standard di programmazione potrebbero implementare questi comandi in modo differente, per esempio chiamando System.out.println al posto di System.out.print, oppure inserendo le parentesi in punti diversi. Per comodità, i commenti inseriti nel codice sono stati tradotti.

```
/*
 * Questo esempio può servire per verificare l'installazione di JDBC.
 * È sufficiente eseguirlo e fornire le informazioni di connessione.
 * Queste selezioneranno "Hello World" dal database.
 */

// Per usare JDBC, è necessario importare il package java.sql
import java.sql.*;

// Importiamo java.io per leggere dalla linea di comando
import java.io.*;

class JdbcCheckup
{
    public static void main (String args [])
        throws SQLException, IOException
    {

```

```
// carica il driver JDBC Oracle
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

// Chiede all'utente le informazioni di connessione
System.out.println ("Please enter information to test connection
to the database");
String user;
String password;
String database;

user = readEntry ("user ");
int slash_index = user.indexOf ('/');
if (slash_index != -1)
{
    password = user.substring (slash_index + 1);
    user = user.substring (0, slash_index);
}
else {
    password = readEntry ("password: ");
}
database = readEntry ("database (a TNSNAME entry): ");

System.out.print ("Connecting to the database...");

System.out.flush ();

System.out.println ("Connecting...");
Connection conn =
    DriverManager.getConnection ("jdbc:oracle:oci8:@" + database,
                               user, password);

System.out.println ("connected.");

// Crea un'istruzione
Statement stmt = conn.createStatement ();

// esegue una query
ResultSet rset=stmt.executeQuery("select 'Hello World' from dual");
// Attraversa ciclicamente i risultati della query
while (rset.next ()) {
    System.out.println (rset.getString (1));
}

System.out.println ("Your JDBC installation is correct.");

// Chiude resultSet
rset.close();

// Chiude l'istruzione
stmt.close();

// Chiude la connessione
conn.close();
}

// Funzione di utilità per leggere una linea dall'input standard
```

```

static String readEntry (String prompt)
{
    try
    {
        StringBuffer buffer = new StringBuffer ();
        System.out.print (prompt);
        System.out.flush ();
        int c = System.in.read ();
        while (c != '\n' && c != -1)
        {
            buffer.append ((char)c);
            c = System.in.read ();
        }
        return buffer.toString ().trim ();
    }
    catch (IOException e)
    {
        return "";
    }
}

```

**NOTA** Le applicazioni di produzione dovrebbero includere molte più sezioni per la gestione delle eccezioni di quelle mostrate in questo esempio breve. Per un esempio sulla gestione delle eccezioni, si veda il Capitolo 34.

Partendo dall'inizio, per prima cosa questo script importa le classi java.sql fornite da Sun Microsystems:

```
import java.sql.*;
```

quindi, viene registrato il driver JDBC fornito da Oracle:

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

Quando vengono richieste delle informazioni, la variabile *user* viene popolata tramite una chiamata alla funzione *readEntry*:

```
user = readEntry ("user: ");
```

La funzione *readEntry* verrà definita più avanti nella classe, nella sezione che inizia con:

```
// Funzione di utilità per leggere una linea dall'input standard
static String readEntry (String prompt)
```

Una volta inseriti i dati di connessione, verrà creato un *oggetto di connessione* per mantenere lo stato della sessione del database. La chiamata a *getConnection* nel listato seguente avvia la connessione:

```
Connection conn =
    DriverManager.getConnection ("jdbc:oracle:oci8:@" + database,
                               user, password);
```

Se durante la connessione si verifica un errore, in genere questo passaggio ne è la causa. Una combinazione nome utente/password errata, un gruppo di driver non corrispondenti o la mancata installazione e configurazione di Oracle Net porteranno al fallimento di questo passaggio.

Per default, un oggetto di connessione viene creato con autocommit attivato, quindi ogni transazione verrà eseguita automaticamente. Per modificare questa impostazione, è possibile utilizzare il comando seguente (partendo dal presupposto che l'oggetto di connessione si chiami *conn*, come mostrato nell'esempio):

```
conn.setAutoCommit(false);
```

La classe crea quindi un'istruzione, esegue una query prestabilita e infine visualizza l'output:

```
// Crea un'istruzione
Statement stmt = conn.createStatement ();
// Esegue una query
ResultSet rset =stmt.executeQuery("select 'Hello World' from dual");
while (rset.next ()) {
    System.out.println (rset.getString (1));
}
```

L'istruzione resultset e la connessione vengono chiuse e il programma completato. Si osserva che nell'esecuzione dell'istruzione sono coinvolti due passaggi: prima, l'istruzione viene creata tramite una chiamata al metodo `createStatement`, quindi viene eseguita attraverso il metodo `executeQuery`. Al posto di `executeQuery` si può utilizzare `execute` o, se si esegue un'istruzione SQL `insert`, `update` o `delete`, `executeUpdate`. Se si seleziona il valore di una colonna (e non una semplice stringa di testo) si dovrebbe specificare il valore di quella colonna nel comando di stampa, come mostrato nel listato seguente:

```
ResultSet rset =stmt.executeQuery("select User from dual");
while (rset.next ()) {
    System.out.println (rset.getString ("USER"));
```

## Uso di JDBC per la manipolazione dei dati

Si provi ad associare gli elementi descritti fino a questo momento: la sintassi di connessione di base, tratta da questo capitolo, e le classi Java del Capitolo 34. L'esempio proposto in questo paragrafo eseguirà una query sulla tabella VALORI\_RAGGIO, calcolerà l'area per ogni valore e inserirà questi valori nella tabella AREE. Pertanto richiede l'uso dell'equivalente Java di un ciclo FOR a cursore, oltre al supporto per i comandi eseguibili e le operazioni di inserimento.

Per l'esempio, si inseriscano tre record nella tabella VALORI\_RAGGIO:

```
delete from VALORI_RAGGIO;
insert into VALORI_RAGGIO(Raggio) values (3);
insert into VALORI_RAGGIO(Raggio) values (4);
insert into VALORI_RAGGIO(Raggio) values (10);
commit;
```

Il listato seguente, *JdbcCircle.java*, contiene i componenti di connessione tratti da *JdbcCheckup.java*. I comandi eseguibili per l'area del cerchio iniziano nella sezione dal nome `RetrieveRadius`, che nel listato compare in grassetto:

```
// Per usare JDBC, è necessario importare il package java.sql
import java.sql.*;
// Importiamo java.io per leggere dalla linea di comando
import java.io.*;
```

```
class JdbcCircle {  
  
    public static void main (String args [])  
        throws SQLException, IOException {  
  
        // Carica il driver JDBC Oracle  
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  
        // Chiede all'utente le informazioni di connessione  
        System.out.println ("Inserire le informazioni per la connessione");  
        String user;  
        String password;  
        String database;  
  
        user = readEntry ("user: ");  
        int slash_index = user.indexOf ('/');  
        if (slash_index != -1) {  
  
            password = user.substring (slash_index + 1);  
            user = user.substring (0, slash_index);  
        }  
        else {  
            password = readEntry ("password: ");  
        }  
        database = readEntry ("database (una voce TNSNAME): ");  
        System.out.println ("Connessione al database...");  
  
        Connection conn =  
            DriverManager.getConnection ("jdbc:oracle:oci8:@" + database,  
                user, password);  
        System.out.println ("connesso.");  
  
        // Crea un'istruzione  
        Statement stmt = conn.createStatement ();  
  
        // RetrieveRadius  
        ResultSet rset =stmt.executeQuery("select Raggio from VALORI_RAGGIO");  
  
        while (rset.next ()) {  
            // Se si intende visualizzare i valori:  
            // System.out.println (rset.getInt ("RAGGIO"));  
            int radInput = rset.getInt ("RAGGIO");  
  
            // Recupera il valore di Raggio, calcola l'area:  
            double result = area(radInput);  
  
            // Inserisce il valore in AREE  
            String sql = "insert into AREE values ("+radInput+","+result+ ")";  
            // Se si intende visualizzare l'istruzione SQL:  
            // System.out.println(sql);  
  
            // Crea un'istruzione per l'inserimento:  
            Statement insArea = conn.createStatement();  
            // Esegue l'inserimento:  
            boolean insertResult = insArea.execute(sql);  
    }  
}
```

```

// Chiude resultSet
rset.close();
// Chiude l'istruzione
stmt.close();
// Chiude la connessione
conn.close();
}

// Funzione di utilità per calcolare l'area
public static double area (int rad) {
    double pi=3.1415927;
    double areacircle=pi*rad*rad;
    return areacircle;
}

// Funzione di utilità per leggere una linea dall'input standard
static String readEntry (String prompt) {

    try {

        StringBuffer buffer = new StringBuffer ();
        System.out.print (prompt);
        System.out.flush ();
        int c = System.in.read ();
        while (c != '\n' && c != -1) {

            buffer.append ((char)c);
            c = System.in.read ();
        }
        return buffer.toString ().trim ();
    }
    catch (IOException e) {

        return "";
    }
}
}

```

Nella sezione `RetrieveRadius`, la query da eseguire viene passata al metodo `executeQuery`. Il risultato della query viene memorizzato in una variabile di nome `rset`.

```
// RetrieveRadius
ResultSet rset =stmt.executeQuery("select Raggio from VALORI_RAGGIO");
```

Come sottolineato nel Capitolo 34, in Java è possibile ricorrere a un ciclo WHILE per avvicinarsi alla funzionalità di un ciclo FOR a cursore PL/SQL. La parte seguente della sezione `RetrieveRadius` utilizza il metodo `getInt` per recuperare il valore Raggio intero dal gruppo dei risultati. Se il valore fosse stato una stringa, sarebbe stato impiegato il metodo `getString`.

```

while (rset.next ()) {
    // Se si intende visualizzare i valori:
    // System.out.println (rset.getInt ("RAGGIO"));
    int radInput = rset.getInt ("RAGGIO");
}
```

La variabile *radInput*, assegnata nel listato precedente, ora può essere utilizzata come input nel calcolo dell'area. Il calcolo dell'area viene eseguito tramite una funzione di utilità (fare riferimento al listato JdbcCircle completo) basata sugli esempi di codice presentati nel Capitolo 34.

```
// Recupera il valore Raggio, calcola l'area:  
double result = area(radInput);
```

Ora si conosce il valore del raggio (la variabile *radInput*) e il valore dell'area (la variabile *result*) da usare quando si popola la tabella AREE. Si crei l'istruzione insert per eseguire l'inserimento. Innanzitutto, si utilizzino le funzioni di concatenazione di stringhe in Java per concatenare il testo con i valori delle variabili. Si presti attenzione a non inserire un punto e virgola alla fine della stringa di comandi generata.

```
// Inserisce il valore in AREE  
String sql = "inserisce nei valori di AREE (" + radInput + "," + result + ")";  
// Se si intende visualizzare l'istruzione SQL:  
// System.out.println(sql);
```

Successivamente, si crei un'istruzione e si esegua il comando insert (tramite una chiamata alla variabile *sql* che contiene il testo del comando):

```
// Crea un'istruzione per l'inserimento:  
Statement insArea = conn.createStatement();  
// Esegue l'inserimento:  
boolean insertResult = insArea.execute(sql);
```

Ha funzionato? Si compili e si esegua il file JdbcCircle.java, quindi si verifichi la tabella AREE:

```
select * from AREE  
order by Raggio;
```

RAGGIO	AREA
3	28.27
4	50.27
10	314.16

Questo esempio si basa su un calcolo semplice, l'area di un cerchio, tuttavia illustra i concetti chiave nello sviluppo JDBC: connessione al database, recupero dei dati, implementazione di un ciclo FOR a cursore, assegnamento delle variabili, creazione di un'istruzione SQL dinamica ed esecuzione di DML. Nel prossimo paragrafo si riprenderà questo esempio nella sua versione SQLJ per illustrare le differenze di funzionalità e codifica esistenti tra i due linguaggi.

### 35.3 SQLJ

SQLJ è un preprocessore che incorpora i comandi SQL sotto forma di chiamate JDBC all'interno di un programma Java. A differenza di JDBC, SQLJ effettua il controllo della sintassi sulle istruzioni SQL durante il processo di compilazione. Nel prossimo paragrafo si imparerà a configurare SQLJ. SQLJ richiede l'utilizzo di un altro insieme di classi, nel file translator.zip, per generare delle classi Java.

## Passaggi di configurazione aggiuntivi per SQLJ

Se si accede a Oracle9i con SQLJ, si dovrà impostare la variabile di sistema CLASSPATH in modo che includa il file translator.zip (per tutte le versioni di JDK) fornito da Oracle. Questo file è situato nella directory /sqlj/lib sotto la directory home di Oracle.

**NOTA** *Questo passaggio non viene eseguito dall'installazione standard di Oracle9i, quindi lo si dovrà effettuare manualmente. Se le variabili di sistema non sono impostate correttamente, non si potranno compilare le classi Java che utilizzano le classi SQLJ di Oracle.*

Per impostare le variabili di sistema, fare clic sull'icona Sistema nel Pannello di controllo. Scegliere la scheda Ambiente per elencare le variabili ambientali e le loro definizioni. Se i file binari JDK non sono ancora stati aggiunti all'impostazione PATH (come mostrato in precedenza in questo capitolo), è necessario aggiungerle prima di utilizzare SQLJ.

**NOTA** *Anche il programma SQL, sqlj.exe, dev'essere presente nel percorso. Per default, Oracle installerà questo programma nella directory /bin, sotto la directory home di Oracle.*

Sostituire "E:\Oracle\Ora91" con la directory home del software Oracle, e sostituire "E:\jdk131" con la directory in cui è stato scaricato JDK. La variabile ambientale CLASSPATH per gli sviluppatori SQLJ è diversa da quella utilizzata dagli sviluppatori JDBC. Per lo sviluppo JDBC, l'unico file che dev'essere contenuto nel percorso delle classi è classes12.zip.

Per SQLJ si dovrà aggiungere la directory dei file translator.zip e runtime12.zip SQLJ, in genere la directory /sqlj/lib sotto la directory home del software Oracle. Nel listato seguente, l'impostazione CLASSPATH a una linea è mostrata su più linee:

```
.;E:\Oracle\Ora91\jdbc\lib\classes12.zip;
E:\Oracle\Ora91\sqlj\lib\runtime12.zip;
E:\Oracle\Ora91\sqlj\lib\translator.zip
```

**NOTA** *SostituiRE "E:\Oracle\Ora91" con la directory home del software Oracle.*

Se durante la compilazione degli esempi SQLJ di questo capitolo si incontrano degli errori, si verifichi la variabile ambientale CLASSPATH. Potrebbe essere necessario chiudere e riavviare la finestra dei comandi perché le modifiche apportate alle impostazioni CLASSPATH abbiano effetto.

**NOTA** *È necessario utilizzare una versione di JDK compatibile con la release di Oracle impiegata. Se si usa una nuova release di JDK con una release più datata dei driver di Oracle, si potrebbe andare incontro a errori di "violazione di accesso" durante l'esecuzione dei programmi.*

## Verifica della configurazione di SQLJ

Oracle mette a disposizione uno script di prova chiamato TestInstallSQLJ.sqlj nella directory /sqlj/demo, situata sotto la directory home del software Oracle. TestInstallSQLJ.sqlj a sua volta legge un file di nome connect.properties collocato nella stessa directory. Quest'ultimo file identifica il database, il nome utente e la password da utilizzare durante la connessione dimostrativa. Per esempio, il file connect.properties potrebbe contenere le voci seguenti:

```
sqlj.url=jdbc:oracle:oci8:@
sqlj.user=scott
sqlj.password=tiger
```

**NOTA** Aggiungere il nome del proprio database subito dopo il simbolo @ nella voce sqlj.url e utilizzare il nome utente e la password di prova al posto dei valori di esempio del precedente listato.

Il file dimostrativo TestInstallSQLJ.sqlj inserisce una riga in una tabella chiamata SALES, che presenta una colonna di nome Item\_Name. Ai fini di questa dimostrazione, è possibile creare la tabella SALES con il comando seguente:

```
create table SALES
(Item_Number      NUMBER,
Item_Name        CHAR(30),
Sales_Date       DATE,
Cost             NUMBER,
Sales_Rep_Number NUMBER,
Sales_Rep_Name   CHAR(20));
```

Se si possiede già un account SCOTT/TIGER nel database, lo si potrà utilizzare, insieme alla sua tabella SALES, per la prova di connessione (questa operazione però cancellerà le righe contenute in questa tabella). Se si preferisce ricorrere a JDBC per la creazione della tabella, si può eseguire la classe TestInstallCreateTable.java, fornita nella directory /sqlj/demo, per creare la tabella SALES. Il file TestInstallCreateTable.java si basa sulle informazioni di connessione specificate nel file connect.properties.

Dopo aver creato la tabella SALES e aver configurato correttamente il file connect.properties, si traduca il file TestInstallSQLJ.sqlj:

```
sqlj TestInstallSQLJ.sqlj
```

Questo passaggio dovrebbe essere completato senza errori. In quanto parte della traduzione SQLJ, la classe Java verrà compilata, e quindi si potrà eseguire:

```
java TestInstallSQLJ
```

Se la configurazione è corretta, dovrebbe comparire l'output seguente:

```
Hello, SQLJ!
```

**NOTA** La parte più lunga della fase di traduzione è la compilazione Java. Se il processo di traduzione genera un file JAVA, si potrà compilare questo file manualmente (tramite javac). In base agli errori incontrati durante la fase di compilazione, la compilazione manuale del file potrebbe fornire informazioni di debug più importanti di quelle offerte dal traduttore.

## 35.4 Uso delle classi SQLJ

Durante la traduzione del file TestInstallSQLJ.sqlj sono stati creati numerosi file nuovi, tra cui un file TestInstallSQL.java e un file compilato TestInstallSQLJ.class. Avendo già analizzato un

file JDBC in questo capitolo, è utile dare un'occhiata al file TestInstallSQLJ.java per vedere cosa ha fatto il traduttore. Si noterà che ha creato una classe chiamata MyIter:

```
class MyIter
extends sqlj.runtime.ref.ResultSetIterImpl
implements sqlj.runtime.NamedIterator
{
    public MyIter(sqlj.runtime.profile.RTResultSet resultSet)
        throws java.sql.SQLException
    {
        super(resultSet);
        ITEM_NAMEx = findColumn("ITEM_NAME");
    }
    public String ITEM_NAME()
        throws java.sql.SQLException
    {
        return resultSet.getString(ITEM_NAMEx);
    }
    private int ITEM_NAMEx;
}
```

La classe MyIter è un iteratore SQLJ, una struttura utilizzata per iterare tra i risultati di query che restituiscono più righe. Più avanti nel file, l'iteratore servirà come base per la query:

```
MyIter iter;
// #sql iter = { SELECT ITEM_NAMEFROM SALES };
```

La query viene così eseguita:

```
sqlj.runtime.profile.RTResultSet __sJT_result =
__sJT_execCtx.executeQuery();
iter = new MyIter(__sJT_result);
```

SQLJ visualizza i dettagli JDBC; non si dovranno inserire manualmente tutto il codice di gestione delle query. Per ogni istruzione SQL, SQLJ crea il codice necessario a eseguirle tramite JDBC e Java. Innanzitutto, crea un oggetto *contesto di connessione*. Il contesto di connessione mantiene la connessione JDBC. Ogni istruzione eseguita usa un oggetto contesto di connessione differente. Per esempio, il file TestInstallSQLJ.java contiene il comando seguente per creare un contesto di connessione:

```
sqlj.runtime.ConnectionContext __sJT_connCtx =
sqlj.runtime.ref.DefaultContext.getDefaultContext();
```

Nel comando precedente viene creato un oggetto contesto di connessione che si basa sulle opzioni di contesto predefinite. Un contesto predefinito supporta una connessione che le classi possono usare per accedere al database senza dover specificare un oggetto contesto di connessione.

Nel contesto di connessione, tutti i comandi vengono eseguiti tramite gli oggetti *contesto di esecuzione*. In TestInstallSQLJ.java, dovrebbero esserci comandi simili ai seguenti che servono a creare un contesto di esecuzione:

```
sqlj.runtime.ExecutionContext __sJT_execCtx =
__sJT_connCtx.getExecutionContext();
if (__sJT_execCtx == null)
    sqlj.runtime.error.RuntimeRefErrors.raise_NULL_EXEC_CTX();
```

```

synchronized (_sJT_execCtx) {
    sqlj.runtime.profile.RTStatement _sJT_stmt =
    _sJT_execCtx.registerStatement(_sJT_connCtx,
        TestInstallSQLJ_SJProfileKeys.getKey(0), 0);
}

```

L'ultima linea del listato precedente si riferisce alle *chiavi di profilo*. Quando è stato tradotto il file TestInstallSQLJ.sqlj, Oracle ha creato automaticamente diversi file, oltre al file TestInstallSQLJ.java. Il traduttore recupera le istruzioni SQL dal file SQLJ creato e crea altri due file: uno con l'estensione .ser, che memorizza i profili dell'istruzione, e un file con un'estensione .class che memorizza le chiavi di profilo. Pertanto, quando si legge il file TestInstallSQLJ.java, si noterà che i comandi SQL sono stati racchiusi in commenti, come mostrato nel listato seguente:

```
// #sql iter = { SELECT ITEM_NAME FROM SALES };
```

I comandi SQL effettivamente eseguiti dal programma non sono visibili, a meno che non si possieda il codice sorgente (il file .sqlj oppure il file .java creati dal traduttore).

## Uso di SQLJ per la manipolazione dei dati

Se si utilizza TestInstallSQLJ.sqlj come base, è possibile sviluppare un programma che recupera i dati dalla tabella VALORI\_RAGGIO esegue il calcolo dell'area di un cerchio e inserisce i risultati nella tabella AREE. Dato che SQLJ si occupa della gestione della connessione JDBC, il flusso logico del programma SQLJ dovrebbe essere più semplice da seguire rispetto al programma JDBC, per quelle persone che hanno già una certa familiarità con PL/SQL.

Il listato seguente è il file SqljCircle.sqlj. Come nell'esempio di JdbcCircle.java, riportato precedentemente in questo capitolo, il calcolo dell'area viene effettuato tramite una funzione di utilità. La parte più importante di questo esempio, ossia i comandi necessari per estrarre e manipolare i dati, è evidenziata in grassetto.

```

/*
 * Importa la classe SQLException. SQLException deriva da
 * JDBC. La clausole #sql eseguibili comportano delle chiamate a JDBC,
 * quindi i metodi contenenti clausole #sql eseguibili
 * dovranno catturare oppure lanciare SQLException.
 */
import java.sql.SQLException ;
import oracle.sqlj.runtime.Oracle;

// iteratore per la selezione
#sql iterator MyIter (int Raggio);

class SqljCircle {

    //Metodo Main
    public static void main (String args[]) {

        try {
            /* Se si usa un driver JDBC non Oracle, aggiungere una chiamata a
               DriverManager.registerDriver() per registrare il driver
            */
            // imposta la connessione predefinita all'URL, utente e password
            // specificati nel file connect.properties
            Oracle.connect(SqljCircle.class, "connect.properties");
            SqljCircle ti = new SqljCircle();
            ti.runExample();
        }
    }
}

```

```

} catch (SQLException e) {
    System.err.println("Errore di esecuzione dell'esempio: " + e);
}
} //Fine del metodo main

//Metodo che esegue l'esempio
void runExample() throws SQLException {

    MyIter iter;

    #sql iter = { select Raggio from VALORI_RAGGIO};
    while (iter.next())    {
        // Per visualizzare i valori del raggio:
        //System.out.println(iter.Raggio());

        int radInput = iter.Raggio();
        double result=area(radInput);
        // Per visualizzare i valori del risultato:
        //System.out.println(result);

        #sql { insert into AREE values(:radInput,:result)};
        #sql {commit};
    }
    iter.close();
}

// Funzione di utilità per calcolare l'area
public static double area (int rad) {
    double pi=3.1415927;
    double areacircle=pi*rad*rad;
    return areacircle;
}
}
}

```

Dato che la query su VALORI\_RAGGIO restituirà più righe, all'interno del file SQLJ viene specificato un iteratore. Se si preferisce, è possibile creare e utilizzare i gruppi di risultati JDBC (fare riferimento agli esempi JDBC presentati in precedenza in questo capitolo). Quando si esegue la query per il programma, il risultato verrà passato come valore per l'iteratore:

```
#sql iter = { select Raggio from VALORI_RAGGIO};
```

Quindi, si usa un ciclo WHILE per valutare le righe:

```

while (iter.next())    {
    // Per visualizzare i valori del raggio:
    //System.out.println(iter.Raggio());
}
```

Per ciascuna riga passata all'iteratore, si catturi il valore Raggio e lo si utilizzi come valore *radInput*. Quindi, si crei la variabile *result* per contenere il valore dell'area calcolata, che si basa sul valore Raggio.

```

int radInput = iter.Raggio();
double result=area(radInput);
// Per visualizzare i valori del risultato:
//System.out.println(result);
```

Una volta calcolati, i valori potranno essere inseriti nella tabella AREE. Questo esempio richiede che la tabella AREE sia vuota prima dell'esecuzione della classe SqljCircle. Nel comando `insert`, le variabili `radInput` e `result` sono precedute da un segno di due punti. Si osservi che l'intero comando è racchiuso tra parentesi graffe, {}, e che al loro interno non compare nessun punto e virgola. Ogni riga viene eseguita dopo ciascun inserimento.

```
#sql { insert into AREE values(:radInput,:result)};
#sql {commit};
```

Questo esempio contiene un comando `commit` esplicito in modo che sia possibile aggiungere altri criteri per controllare la frequenza di esecuzione all'interno dell'applicazione.

Ora, si chiuda l'iteratore:

```
iter.close();
```

e l'esempio è terminato. È possibile verificare il risultato nella tabella AREE.

Per l'esempio, si inseriscano tre record nella tabella VALORI\_RAGGIO:

```
delete from AREE;
delete from VALORI_RAGGIO;
insert into VALORI_RAGGIO(Raggio) values (3);
insert into VALORI_RAGGIO(Raggio) values (4);
insert into VALORI_RAGGIO(Raggio) values (10);
commit;
```

Il file `connect.properties` dovrebbe assomigliare al seguente, in cui “orcl” è il nome dell'istanza cui connettersi:

```
sqlj.url=jdbc:oracle:oci8:@orcl
sqlj.user=practice
sqlj.password=practice
```

Il listato seguente mostra la compilazione e l'esecuzione di `SqljCircle.sqlj`:

```
sqlj SqljCircle.sqlj
java SqljCircle
```

È possibile verificare i risultati dall'interno di SQL\*Plus:

```
select * from AREE
order by Raggio;
```

RAGGIO	AREA
3	28.27
4	50.27
10	314.16

Dato che supporta le chiamate SQL senza alcun problema, SQLJ è un linguaggio molto apprezzato dai programmatore JDBC principianti.

Per confronto, il file `SqljCircle.java` creato dal traduttore è tre volte più lungo del file `SqljCircle.sqlj`. Per ulteriori informazioni sulle opzioni disponibili per l'implementazione e la gestione degli iteratori, fare riferimento alla documentazione di Oracle.

Questo capitolo offre un'introduzione generale alle capacità di SQLJ, query, manipolazione dei dati, gestione di gruppi di risultati a più righe, assegnamenti di variabili e operazioni DML. Per ulteriori informazioni sulle capacità di SQLJ, si prega di fare riferimento alla documentazione su SQLJ fornita da Oracle.

## Capitolo 36

# Stored procedure Java

36.1 **Caricamento della classe nel database**

36.2 **Come accedere alla classe**

**E** possibile scrivere stored procedure, trigger, metodi del tipo di oggetto e funzioni che chiamano le classi Java. In questo capitolo verrà introdotto un gruppo di procedure Java di esempio insieme ai comandi che servono a eseguire una procedura. Per apprendere il più possibile da questo capitolo, si dovrebbero conoscere già le strutture Java (Capitolo 34) e i vari impieghi delle classi Java, di JDBC e SQLJ (Capitolo 35). Inoltre, si dovrebbe avere già una certa familiarità con la sintassi di base e l'uso delle stored procedure (fare riferimento al capitolo 29).

Per concentrarsi sulle operazioni di accesso al database delle classi Java, l'esempio di questo capitolo sarà breve. La classe seguente, chiamata AreasDML, contiene dei metodi per inserire e eliminare i record dalla tabella AREE. In questo esempio, non viene effettuato il calcolo dell'area; tutti i valori richiesti vengono forniti durante l'inserimento. Il listato seguente mostra il file AreasDML.sqlj:

```
import sqlj.runtime.*;
import sqlj.runtime.ref.*;
import java.sql.SQLException ;
import oracle.sqlj.runtime.Oracle;
import java.sql.*;

class AreasDML {

    //Metodo insert
    public static void insert (
        oracle.sql.NUMBER  radius,
        oracle.sql.NUMBER  area ) throws SQLException {
        try {
            #sql {insert into AREE values
                  (:radius, :area)  };
        }
        catch (SQLException e) {
            System.err.println(e.getMessage());
        }
    }//Fine del metodo insert

    //Metodo delete
    public static void delete (
        oracle.sql.NUMBER  radius,
        oracle.sql.NUMBER  area ) throws SQLException {
```

```

try {
    #sql {delete from AREE
          where Raggio = :radius};
}
catch (SQLException e) {
    System.err.println(e.getMessage());
}
} //Fine del metodo delete
}

```

Il file AreasDML.sqlj definisce la classe AreasDML. All'interno della classe AreasDML, esistono metodi separati per elaborare gli inserimenti e le cancellazioni. In ciascuna delle specifiche del metodo, sono elencati i tipi di dati per le colonne della tabella AREE:

```

public static void insert (
    oracle.sql.NUMBER radius,
    oracle.sql.NUMBER area )

```

Questi parametri utilizzano i tipi di dati forniti come parte integrante delle librerie di classi di Oracle. Se si usano i tipi di dati Java standard, durante l'esecuzione della classe si potrebbero incontrare problemi legati proprio ai tipi di dati.

**NOTA** *Se si importano le librerie oracle.sqlj.\*, si potrà utilizzare NUMBER al posto di oracle.sql.NUMBER.*

La parte successiva del metodo insert inserisce i valori nella tabella AREE:

```

#sql {insert into AREE values
      (:radius, :area) };

```

Si osservi che i nomi delle variabili distinguono tra maiuscole e minuscole, pertanto devono corrispondere esattamente ai nomi dichiarati precedentemente.

Il metodo delete possiede una struttura molto simile. La sua istruzione SQL è

```

#sql {delete from AREE
      where Raggio = :radius};

```

Si osservi che nessuna delle parti della classe stabilisce una connessione al database, contrariamente a quanto accadeva negli esempi del Capitolo 35. La connessione impiegata per invocare questa classe fornirà il contesto di connessione per i metodi della classe.

Si verifichi la corretta compilazione della classe elaborandola tramite il preprocessore SQLJ:

```
sqlj AreasDML.sqlj
```

Questo passaggio dovrebbe essere completato senza errori. Se durante questa fase di compilazione si verificano degli errori, occorre controllare la configurazione dell'ambiente SQLJ (Capitolo 35).

### 36.1 Caricamento della classe nel database

Prima di caricare una classe nel database, si confermi la sua corretta compilazione attraverso SQLJ. Successivamente, si potrà utilizzare l'utilità loadjava per caricare la classe nel database. La sintassi per l'utilità loadjava è simile alla seguente:

---

```
loadjava {-user | -u} nomeutente/password[@database]
[-opzione_nome -opzione_nome ...] nomefile nomefile ...
```

Le opzioni disponibili per l'utility loadjava sono mostrate nella Tabella 36.1.

Per esempio, per caricare il file AreasDML.sqlj nello schema Practice in un'istanza chiamata "orcl", si può ricorrere al comando seguente:

```
loadjava -u Practice/practice@orcl.world -verbose AreasDML.sqlj
```

Dovrebbe comparire l'output seguente:

```
arguments: '-u' 'practice/practice@orcl.world' '-verbose' 'AreasDML.sql'
created   :JAVA$CLASS$MD5$TABLE
creating  : source AreasDML
created   :CREATE$JAVA$LOB$TABLE
loading   : source AreasDML
creating  : AreasDML
```

**NOTA** *Il nome della classe non dovrebbe corrispondere al nome di nessuno degli oggetti già esistenti nello schema.*

Ora, la classe AreasDML è stata caricata nel database. Ecco cosa è stato creato (da USER\_OBJECTS è possibile effettuare la query su nuovi oggetti basandosi sui loro indicatori orario relativi alla creazione dell'oggetto):

OBJECT_NAME	OBJECT_TYPE
AreasDML	JAVA CLASS
AreasDML	JAVA SOURCE
AreasDML_SJProfileKeys	JAVA CLASS
CREATE\$JAVA\$LOB\$TABLE	TABLE
JAVA\$CLASS\$MD5\$TABLE	TABLE
JAVA\$OPTIONS	TABLE
SYS_C002684	INDEX
SYS_C002685	INDEX
SYS_LOB0000032362C00002\$\$	LOB

9 righe selezionate.

I nomi degli indici e del LOB che iniziano con SYS\_ saranno diversi nel database, poiché ogni nome include un numero di sequenza che è specifico del database. La sorgente e la classe AreasDML e le chiavi di profilo dovranno essere già note dagli esempi SQLJ riportati nel Capitolo 35.

## 36.2 Come accedere alla classe

Ora che la classe AreasDML è stata caricata nel database, si dovrà creare una procedura per chiamare ciascuno dei metodi. Il comando create procedure crea nel listato seguente una procedura chiamata InsertAreaViaJava:

```
create or replace procedure InsertAreaViaJava
(Raggio NUMBER,
```

**Tabella 36.1** Opzioni di loadjava.

OPZIONE	DESCRIZIONE
filenames	Qualunque numero e combinazione di argomenti con nomi di file .java, .class, .sqlj, .ser, .jar, .zip di risorsa, in qualunque ordine.
andresolve	Compila i file sorgente e risolve ciascun file di classe mentre viene caricato.
debug	Genera informazioni di debug.
definer	Specifica che i metodi delle classi caricate verranno eseguiti con i privilegi del loro definitore, non del loro chiamante. Per default, i metodi vengono eseguiti con i privilegi del loro chiamante.
encoding	Imposta (o reimposta) l'opzione <i>-encoding</i> nella tabella del database JAVA\$OPTIONS sul valore specificato.
fileout	Stampa i messaggi su un file di output.
force	Forza il caricamento dei file di classe Java a prescindere dal fatto che siano stati caricati in precedenza. Per default, i file di classe caricati in precedenza vengono rifiutati.
grant	Assegna il privilegio EXECUTE sulle classi caricate agli utenti e ai ruoli elencati.
help	Stampa il testo della guida di loadjava.
javaresource	Indirizza loadjava al caricamento di un file JAR senza scompattarlo.
nousage	Sopprime il messaggio di utilizzo dato se nessuna opzione viene specificata o se viene specificata l'opzione <i>-help</i> .
noserverside	Modifica il comportamento dello strumento loadjava sul lato server in modo che utilizzi un driver JDBC per l'accesso agli oggetti.
noverify	Provoca il caricamento delle classi senza verifica del bytecode. Occorre disporre del privilegio oracle.aurora.security.JServerPermission(Verifier) per eseguire questa opzione. Inoltre questa opzione dev'essere utilizzata in combinazione con <i>-r</i> .
oci8	Indirizza loadjava a comunicare con il database utilizzando il driver JDBC OCI. Questa opzione (predefinita) e <i>-thin</i> sono reciprocamente esclusive.
resolve	Risolve tutti i riferimenti esterni nelle classi caricate e compilate. Se questa opzione non è specificata, i file vengono caricati ma non compilati o risolti fino alla fase di esecuzione.
resolver	Lega gli oggetti dello schema della classe appena creata a uno specificatore di resolver definito dall'utente.
schema	Specifica lo schema per i nuovi oggetti creati.
stdout	Causa il reindirizzamento dell'output a stdout invece che a stderr.
synonym	Crea automaticamente un sinonimo pubblico per la classe caricata.
tableschema	Crea le tabelle interne loadjava in questo schema specificato invece che nello schema di destinazione del file Java.
thin	Indirizza loadjava a comunicare con il database utilizzando il driver JDBC thin. Questa opzione e <i>-oci8</i> si escludono a vicenda.
time	Aggiunge un indicatore orario a tutti i messaggi.
unresolvedok	Quando combinata con <i>-resolve</i> , ignora gli errori non risolti.
user	Specifica un utente, una password e una stringa di connessione al database per il caricamento.
verbose	Attiva la visualizzazione dei messaggi di progresso durante l'esecuzione di loadjava.

```

Area      NUMBER)
as
language java
name 'AreasDML.insert(oracle.sql.NUMBER, oracle.sql.NUMBER)';
/

```

Questa procedura breve ha due parametri: Raggio e Area. La sua clausola language java comunica a Oracle che la procedura sta chiamando un metodo Java. Il nome del metodo è specificato nella clausola name (si verifichi la coerenza con l'uso delle lettere maiuscole). Quindi vengono specificati i formati per la chiamata al metodo Java.

Una procedura separata eliminerà i record della tabella AREA tramite una chiamata al metodo:

```

create or replace procedure DeleteAreaViaJava
(Raggio  NUMBER,
 Area     NUMBER)
as
language java
name 'AreasDML.delete(
    oracle.sql.NUMBER, oracle.sql.NUMBER)';
/

```

Ora, ci si deve spostare in SQL\*Plus e utilizzare il comando call per eseguire le classi Java. Per questo esempio, si inizi con la tabella AREE vuota:

```

delete from AREE;
commit;

call InsertAreaViaJava(10,314.15);

```

Comparirà questa risposta:

Call completed.

Il comando call passerà i valori 10 e 314.15 come parametri alla stored procedure denominata InsertAreaViaJava. Li formatterà secondo le specifiche contenute nella clausola name e li passerà al metodo AreasDML.insert. Questo metodo utilizzerà la connessione corrente per eseguire i propri comandi SQLJ, e la riga verrà inserita. La maggior parte di queste librerie e classi non sarà ancora stata caricata nel database. I passaggi da eseguire sono molti, quindi la prima volta che si esegue la stored procedure Java le prestazioni ne risentiranno, mentre le esecuzioni successive saranno molto più veloci. È possibile verificare l'esito eseguendo una query sulla tabella AREE:

```

select * from AREE;

RAGGIO      AREA
----- -----
10          314.15

```

È possibile eliminare quel record con una chiamata a DeleteAreaViaJava:

```
call DeleteAreaViaJava(10,314.15);
```

Oracle replicherà:

Call completed.

La chiamata a DeleteAreaViaJava verrà compilata molto più rapidamente di quella a InsertAreaViaJava, poiché le librerie Java saranno state caricate durante la prima esecuzione di InsertAreaViaJava.

Si controlli la gestione degli errori passando una stringa di caratteri da inserire:

```
call InsertAreaViaJava('J',314.15);
*
ERRORE at line 1:
ORA-01722: invalid number
```

Come si poteva immaginare, la non corrispondenza dei tipi di dati ha causato un problema: in questo caso Oracle rileva che il dato passato alla procedura PL/SQL non corrisponde al tipo di dato del parametro della procedura.

Quando si effettua il debug delle stored procedure Java, è possibile chiamare System.out.println dall'interno del codice Java, come mostrato nel listato AreasDML.sqlj proposto in precedenza nel capitolo. Per visualizzare l'output nella sessione SQL\*Plus, per prima cosa è necessario eseguire i comandi seguenti:

```
set serveroutput on
call DBMS_JAVA.SET_OUTPUT(10000);
```

Con una chiamata a DBMS\_JAVA.SET\_OUTPUT si reindirizzerà l'output di System.out.println a DBMS\_OUTPUT.PUT\_LINE. Altrimenti, l'output di System.out.println passerà solamente al file di log del database.

## Dove eseguire i comandi

Lo scopo dell'esempio AreasDML è fornire un'introduzione generale all'implementazione delle stored procedure Java. Si supponga di voler rendere più complesse le procedure. Dato che l'area e il raggio sono direttamente legati l'una all'altro, non si dovrà fare altro che accettare il valore del raggio come input per la procedura. Successivamente si potrebbe calcolare l'area e inserire il valore così ottenuto. Come si è visto nei Capitoli 27, 34 e 35, è possibile eseguire questi calcoli in Java oppure in PL/SQL.

La scelta di una piattaforma e di un linguaggio per l'elaborazione dei dati influirà sulle prestazioni. In linea di massima, la prima volta che si esegue una stored procedure Java (mentre le classi vengono caricate) si riscontrerà un calo delle prestazioni, tuttavia le esecuzioni successive non subiranno la stessa sorte. Se si eseguono ripetutamente gli stessi comandi, il calo di prestazioni durante la prima esecuzione sarà minimo.

Per esempio, più esecuzioni della procedura InsertAreaViaJava verrebbero completate in 0,5 secondi, 0,2 secondi, 0,2 secondi e 0,2 secondi. Una serie di inserimenti nella tabella AREE verrebbe completata in una media di 0,18 secondi.

La disconnessione e la riconnessione influiscono sulle prestazioni, ma non così tanto come la prima esecuzione della procedura. Quando si utilizzano le stored procedure Java, è quindi necessario essere consapevoli del modo in cui l'applicazione interagisce con Oracle. Se gli utenti si disconnettono e riconnettono di frequente, la prima volta che eseguiranno la procedura dopo ogni connessione riscontreranno un calo delle prestazioni. Inoltre, la prima esecuzione della procedura che avviene dopo l'avvio del database subirà il calo di prestazioni più sensibile. Du-

rante la fase di progettazione dello sviluppo dell'applicazione, si dovranno prendere in considerazione gli influssi che queste azioni esercitano sulle prestazioni.

Ecco come ridurre i cali di prestazioni associati alle chiamate alle stored procedure Java.

- Eseguire ogni stored procedure Java subito dopo l'avvio del database. Nel caso dell'esempio di AreasDML, questo potrebbe richiedere l'inserimento e la cancellazione di un record di prova dopo ogni avvio.
- Ridurre il numero delle connessioni al database di breve durata; per esempio quelle effettuate per connection pooling.
- Eseguire la maggior parte delle manipolazioni dirette del database tramite SQL e PL/SQL standard.

Java è una tecnologia adatta per essere impiegata in molti processi di calcolo e per visualizzare le manipolazioni. In genere, SQL e PL/SQL sono leggermente più efficienti di Java per quanto riguarda le connessioni al database e la manipolazione dei dati. Tuttavia, le prestazioni di Java in Oracle lo rendono un'alternativa valida a PL/SQL; la causa principale dei cali di prestazioni per le stored procedure Java è il tempo necessario per eseguire il wrapper PL/SQL. Quando si progettano applicazioni che integrano Java, SQL e PL/SQL per ciascun linguaggio si dovranno considerare principalmente le funzioni che esso esegue al meglio.

Se il codice richiede molti cicli e manipolazioni di dati, Java dovrebbe offrire prestazioni migliori di PL/SQL. Gli svantaggi derivati dall'uso di Java per la creazione di oggetti Java e l'interazione dell'intestazione PL/SQL verranno controbilanciati dalle migliori prestazioni che Java raggiunge durante il lavoro effettivo.

Il calcolo dell'area di un cerchio è pura aritmetica che PL/SQL e Java gestiscono facilmente. Pertanto, il calcolo dell'area potrebbe essere inserito in qualunque parte dell'applicazione, in una procedura PL/SQL che calcola l'area prima di passarla a SQLJ per essere inserita, oppure come parte della classe SQLJ. In linea di massima, PL/SQL e SQL gestiscono meglio le azioni che richiedono la selezione dei dati dal database e la loro manipolazione. Java potrebbe invece eseguire più rapidamente i calcoli sui dati selezionati. Durante la progettazione dell'applicazione, occorre accertarsi di utilizzare lo strumento giusto per la funzione adatta in modo da evitare inutili cali di prestazioni.



Parte sesta

## **GUIDE**



## Capitolo 37

# Guida al dizionario dati di Oracle9i

- 37.1 **Nota sulla nomenclatura**
- 37.2 **Le “carte stradali”:  
DICTIONARY (DICT) e DICT\_COLUMNS**
- 37.3 **Oggetti da cui è possibile selezionare:  
tabelle (e colonne), viste, sinonimi e sequenze**
- 37.4 **Vincoli e commenti**
- 37.5 **Indici e cluster**
- 37.6 **Tipi di dati astratti, strutture correlate  
a ORDBMS e LOB**
- 37.7 **Database link e viste materializzate**
- 37.8 **Trigger, procedure, funzioni e package**
- 37.9 **Dimensioni**
- 37.10 **Allocazione e uso dello spazio, comprese  
partizioni e partizioni secondarie**
- 37.11 **Utenti e privilegi**
- 37.12 **Ruoli**
- 37.13 **Audit**
- 37.14 **Varie**
- 37.15 **Monitoraggio: le tabelle  
di prestazioni dinamiche V\$**

I dizionario dati di Oracle memorizza tutte le informazioni che servono a gestire gli oggetti nel database. Anche se costituisce abitualmente il dominio degli amministratori di database (DBA), il dizionario dati è anche una fonte di informazioni preziose per gli sviluppatori e gli utenti finali.

In questo capitolo, il dizionario dati, ovvero la directory interna di Oracle, viene descritto dal punto di vista dell’utente finale; per questo motivo, le tabelle e le viste del dizionario dati non sono elencate in ordine alfabetico, ma raggruppate per funzione (tabelle, sicurezza e così via). Questo tipo di organizzazione consente di individuare rapidamente le informazioni desiderate. Sono descritte tutte le viste del dizionario dati più utilizzate, con esempi del loro impiego.

In base alle opzioni di configurazione di Oracle utilizzate, alcuni dei gruppi non riguardano tutti i database. Le viste più utilizzate sono analizzate per prime.

**NOTA** *Data la continua introduzione di nuove caratteristiche, è molto probabile che il dizionario dati venga modificato con ogni futura release di Oracle.*

### 37.1 Nota sulla nomenclatura

Con alcune eccezioni, i nomi degli oggetti nel dizionario dati di Oracle iniziano con uno dei tre prefissi “USER”, “ALL” o “DBA”. In genere, i record nelle viste “USER” contengono informazioni relative a oggetti posseduti dall’account che effettua la query. I record nelle viste “ALL” comprendono i record “USER”, oltre a informazioni sugli oggetti per i quali sono stati concessi privilegi al pubblico (PUBLIC) o all’utente. Le viste “DBA” comprendono tutti gli oggetti del database, a prescindere dal loro proprietario. Le viste “USER”, “ALL” e “DBA” sono disponibili per la maggior parte degli oggetti del database.

Dal momento che per scelta questa guida tende a privilegiare l’utente, si presterà particolare attenzione alle viste “USER” o a quelle accessibili a tutti gli utenti. Le viste “ALL” e “DBA” sono descritte solo quando risultano applicabili.

### 37.2 Le “carte stradali”: DICTIONARY (DICT) e DICT\_COLUMNS

Alle descrizioni degli oggetti che costituiscono il dizionario dati di Oracle è possibile accedere tramite una vista di nome DICTIONARY. Questa vista, accessibile anche attraverso il sinonimo pubblico DICT, esegue una query sul database per stabilire quali viste del dizionario dati l’utente può visualizzare. Inoltre, cerca dei sinonimi pubblici per queste viste.

Nell’esempio seguente si effettua una query su DICT per ottenere i nomi di tutte le viste del dizionario dati i cui nomi includano la stringa ‘VIEWS’. La selezione da DICT tramite un account non DBA, come mostrato nel listato seguente, restituisce il nome dell’oggetto e i commenti per ciascun oggetto del dizionario dati che soddisfa i criteri di ricerca. Nella vista sono presenti solo due colonne: Table\_Name e i commenti associati alla tabella (Comments).

```
column Comments format a35 word_wrapped
column Table_Name format a25

select Table_Name, Comments
  from DICT
 where Table_Name like '%VIEWS%';



| TABLE_NAME             | COMMENTS                                                                               |
|------------------------|----------------------------------------------------------------------------------------|
| ALL_BASE_TABLE_MVIEWS  | All materialized views with log(s)<br>in the database that the user can<br>see         |
| ALL_MVIEWS             | All materialized views in the<br>database                                              |
| ALL_REGISTERED_MVIEWS  | Remote materialized views of local<br>tables that the user can see                     |
| ALL_VIEWS              | Description of views accessible to<br>the user                                         |
| USER_BASE_TABLE_MVIEWS | All materialized views with log(s)<br>owned by the user in the database                |
| USER_MVIEWS            | All materialized views in the<br>database                                              |
| USER_REGISTERED_MVIEWS | Remote materialized views of local<br>tables currently using logs owned<br>by the user |
| USER_VIEWS             | Description of the user's own views                                                    |


```

È possibile eseguire una query sulle colonne delle viste del dizionario tramite la vista DICT\_COLUMNS. Come la vista DICTIONARY, anche DICT\_COLUMNS visualizza i commenti inseriti nel database per le viste del dizionario dati. DICT\_COLUMNS ha tre colonne: Table\_Name, Column\_Name e Comments. Con una query su DICT\_COLUMNS è possibile determinare quali viste del dizionario dati sono più utili per le proprie esigenze.

Per esempio, se si desidera visualizzare le informazioni relative all'uso e all'allocazione dello spazio per gli oggetti del database, ma non si conosce con esattezza quali viste del dizionario dati memorizzino queste informazioni, è possibile eseguire una query su DICT\_COLUMNS, come mostrato nell'esempio seguente, che cerca tutte le tabelle del dizionario che possiedono una colonna di nome Blocchi:

```
select Table_Name
  from DICT_COLUMNS
 where Column_Name = 'BLOCCHI'
   and Table_Name like 'USER%';
```

TABLE_NAME
-----
USER_ALL_TABLES
USER_EXTENTS
USER_FREE_SPACE
USER_OBJECT_TABLES
USER_SEGMENTS
USER_TABLES
USER_TAB_PARTITIONS
USER_TAB_SUBPARTITIONS
USER_TS_QUOTAS

Per elencare tutti i nomi delle colonne disponibili che possono essere stati utilizzati nell'esempio precedente, occorre effettuare una query su DICT\_COLUMNS:

```
select distinct Column_Name
  from DICT_COLUMNS
 order by Column_Name;
```

Ogni volta che non si sa con sicurezza dove trovare i dati desiderati, è sufficiente consultare DICTIONARY e DICT\_COLUMNS. Se si scopre che molte viste potrebbero essere utili, occorre effettuare una query su DICTIONARY (come nel primo esempio) per visualizzare i commenti relativi a ciascuna vista.

### 37.3 Oggetti da cui è possibile selezionare: tabelle (e colonne), viste, sinonimi e sequenze

Il catalogo di un utente elenca tutti gli oggetti da cui può selezionare dei record, ovvero qualsiasi oggetto che possa essere menzionato nella clausola from di una query. Anche se nelle clausole from non si fa riferimento diretto alle sequenze, Oracle le comprende nel catalogo. In questo paragrafo si spiega come recuperare informazioni sulle tabelle, colonne, viste, sinonimi, sequenze e catalogo utente.

## Catalogo: USER\_CATALOG (CAT)

Con una query su USER\_CATALOG è possibile visualizzare tutte le tabelle, le viste, i sinonimi e le sequenze di cui l'utente è proprietario. La colonna Table\_Name visualizza il nome dell'oggetto (anche se questo non è una tabella), mentre la colonna Table\_Type ne visualizza il tipo:

```
select Table_Name, Table_Type
  from USER_CATALOG
 where Table_Name like 'T%';
```

Il riferimento alla vista USER\_CATALOG può avvenire anche tramite il sinonimo pubblico CAT.

Esistono altri due cataloghi. ALL\_CATALOG elenca tutto quello che è contenuto nella vista USER\_CATALOG, più qualunque oggetto per il quale sia stato concesso l'accesso all'utente o al pubblico (PUBLIC). DBA\_CATALOG è una versione a livello di DBA del catalogo in cui sono visualizzate tutte le tabelle, le viste, le sequenze e i sinonimi presenti nel database. Oltre alle colonne Table\_Name e Table\_Type, visualizzate nella query di USER\_CATALOG, ALL\_CATALOG e DBA\_CATALOG includono una colonna Owner.

## Oggetti: USER\_OBJECTS (OBJ)

USER\_CATALOG visualizza solamente le informazioni relative a tabelle, viste, sequenze e sinonimi. Per recuperare informazioni su tutti i tipi di oggetti, occorre effettuare una query su USER\_OBJECTS. Questa vista può servire anche per scoprire quanti tipi di oggetti esistono, compresi cluster, database link, directory, funzioni, indici, librerie, package, corpi di package, classi Java, tipi di dati astratti, piani delle risorse, sequenze, sinonimi, tabelle, trigger, viste materializzate, LOB e viste. Le colonne di USER\_OBJECTS sono elencate nella Tabella 37.1.

USER\_OBJECTS (nota anche con il suo sinonimo pubblico OBJ) contiene numerose informazioni vitali che non sono presenti in altre viste del dizionario dati. In essa sono registrate le date di creazione (colonna Created) e di ultima modifica degli oggetti (colonna Last\_DDL\_Time). Queste colonne sono molto utili quando si tenta di riconciliare diversi gruppi di oggetti nella stessa applicazione.

**NOTA** *Se si ricreano oggetti con qualsiasi metodo, per esempio tramite l'utility ImportS (si rimanda al Capitolo 40), i relativi valori di Created cambieranno, indicando la data di ultima creazione.*

Sono disponibili altri due elenchi di oggetti. ALL\_OBJECTS elenca tutto ciò che si trova nella vista USER\_OBJECTS oltre agli oggetti per i quali è stato concesso l'accesso all'utente o al pubblico (PUBLIC). DBA\_OBJECTS è una versione a livello di DBA dell'elenco di oggetti, che visualizza tutti gli oggetti del database. Oltre alle colonne mostrate nella vista USER\_OBJECTS, ALL\_OBJECTS e DBA\_OBJECTS comprendono una colonna Owner.

## Tabelle: USER\_TABLES (TABS)

Anche se tutti gli oggetti di un utente sono visualizzati in USER\_OBJECTS, pochi sono gli attributi di tali oggetti ivi visualizzati. Per ottenere più informazioni su un oggetto, è necessario visualizzare la vista specifica per quel tipo di oggetto. Per le tabelle, la vista è USER\_TABLES. È possibile fare riferimento a questa vista anche tramite il sinonimo pubblico TABS.

**Tabella 37.1** Colonne di USER\_OBJECTS.

NOME COLONNA	DESCRIZIONE
Object_Name	Il nome dell'oggetto.
SubObject_Name	Il nome del "sottooggetto", come una partizione.
Object_ID	Un numero ID unico assegnato da ORACLE all'oggetto.
Data_Object_ID	ID oggetto del segmento che contiene i dati dell'oggetto.
Object_Type	Tipo dell'oggetto ('TABLE', 'INDEX', 'TABLE PARTITION' e così via).
Created	Data di creazione dell'oggetto (una colonna DATE).
Last_DDL_Time	Data dell'ultimo comando DDL utilizzato sull'oggetto, compresi alter, grant e revoke.
Timestamp	Data di creazione degli oggetti (uguale a Created, ma memorizzata come una colonna alfanumerica).
Status	Stato dell'oggetto ('VALID' o 'INVALID').
Temporary	Flag che indica se l'oggetto è una tabella temporanea.
Generated	Flag che indica se il nome dell'oggetto è stato generato dal sistema.
Secondary	Flag che indica se l'oggetto è un indice secondario creato da un indice di dominio.

**NOTA** Le versioni precedenti di Oracle comprendevano una vista di nome TAB. Questa vista, le cui funzioni sono simili a TABS, è ancora supportata perché esistono prodotti di Oracle che fanno riferimento a essa. Tuttavia, TAB non contiene le colonne contenute da TABS. Nelle query sul dizionario dati occorre utilizzare TABS.

Le colonne in USER\_TABLES possono esse suddivise in quattro categorie principali ("Identificazione", "Correlate allo spazio", "Correlate alle statistiche" e "Altre"), come illustrato nella Tabella 37.2.

Il nome della tabella compare nella colonna Table\_Name. La colonna Backup mostra se è stato eseguito o meno un backup della tabella dopo la sua ultima modifica. La colonna Partitioned contiene il valore 'Y' se la tabella è stata partizionata (le viste del dizionario dati correlate alle partizioni sono descritte nel paragrafo "Allocazione e utilizzo dello spazio" di questo capitolo). Nel caso di una tabella di oggetti, la colonna Table\_Type\_Owner visualizza il proprietario del tipo.

Le colonne "correlate allo spazio" sono descritte alla voce Memorizzazione del Capitolo 42. Le colonne "correlate alle statistiche" vengono popolate quando si analizza la tabella.

Con una query sulla colonna Table\_Name di USER\_TABLES vengono visualizzati i nomi di tutte le tabelle nell'account corrente. Nella query seguente, sono elencate tutte le tabelle i cui nomi iniziano con a lettera 'L'.

```
select Table_Name from USER_TABLES
where Table_Name like 'L%';
```

La vista ALL\_TABLES mostra tutte le tabelle di proprietà dell'utente e quelle per le quali gli è stato concesso l'accesso. La maggior parte degli strumenti di generazione di report esterni,

che elencano le tabelle disponibili per le query, ottiene tale elenco con una query su ALL\_TABLES. Dato che questa vista è in grado di contenere voci per più utenti, contiene, oltre alle colonne elencate nella Tabella 37.2, una colonna Owner, DBA\_TABLES, che elenca tutte le tabelle del database, ha le stesse definizioni delle colonne di ALL\_TABLES.

Le colonne Degree e Instances nella categoria “Altre” della Tabella 37.2 fanno riferimento a come viene effettuata la query sulla tabella durante le query parallele. Per ulteriori informazioni su Degree, Instances e gli altri parametri delle tabelle, si consulti la voce relativa al comando create table nel Capitolo 42.

**NOTA** *Per le tabelle esterne, fare riferimento a USER\_EXTERNAL\_TABLES.*

## Colonne: USER\_TAB\_COLUMNS (COLS)

Anche se gli utenti non eseguono query dalle colonne, la vista del dizionario dati che visualizza le colonne è strettamente legata alla vista del dizionario dati delle tabelle. Questa vista, di nome USER\_TAB\_COLUMNS, elenca informazioni specifiche sulle colonne. La query su USER\_TAB\_COLUMNS può essere eseguita anche tramite il sinonimo pubblico COLS.

Le colonne su cui è possibile effettuare una query da USER\_TAB\_COLUMNS possono essere suddivise in tre categorie principali, come riportato nella Tabella 37.3.

Le colonne Table\_Name e Column\_Name contengono i nomi delle tabelle e colonne. L’uso delle colonne “correlate alla definizione” è descritto nella voce “Tipi di dati” del Capitolo 42. Le colonne “correlate alle statistiche” sono popolate quando la tabella viene analizzata (si veda il

**Tabella 37.2** Colonne di USER\_TABLES.

IDENTIFICAZIONE	CORRELATE ALLO SPAZIO	CORRELATE ALLE STATISTICHE	ALTRÉ
Table_Name	Tablespace_Name	Num_Rows	Degree
Backed_Up	Cluster_Name	Blocks	Instances
Partitioned	Pct_Free	Empty_Blocks	Cache
IOT_Name	Pct_Used	Avg_Space	Table_Lock
Logging	Ini_Trans	Chain_Cnt	Buffer_Pool
IOT_Type	Max_Trans	Avg_Row_Len	Row_Movement
Temporary	Initial_Extent	Sample_Size	Duration
Nested	Next_Extent	Last_Analyzed	Skip_Corrupt
Secondary	Min_Exents	Avg_Space_Freelist_Blocks	Monitoring
	Max_Exents	Num_Freelist_Blocks	Cluster_Owner
	Pct_Increase	Global_Stats	Dependencies
	Freelists	User_Stats	
	Freelist_Groups		

comando `analyze` nel Capitolo 42). Le statistiche relative alle colonne sono fornite anche nella vista `USER_TAB_COL_STATISTICS` (descritta tra breve).

Per visualizzare le definizioni delle colonne per una tabella, occorre effettuare una query su `USER_TAB_COLUMNS` specificando `Table_Name` nella clausola `where`:

```
select Column_Name, Data_Type
  from USER_TAB_COLUMNS
 where Table_Name = 'RIVISTA';
```

COLUMN_NAME	DATA_TYPE
ARGOMENTO	VARCHAR2
SEZIONE	CHAR
PAGINA	NUMBER

Le informazioni di questo esempio si possono ottenere anche con il comando SQLPLUS `describe`. Tuttavia, questo comando non offre la possibilità di visualizzare le impostazioni predefinite e le statistiche della colonna. La vista `ALL_TAB_COLUMNS` visualizza le colonne di tutte le tabelle e viste possedute dall'utente, e di tutte le tabelle e viste per le quali all'utente è concesso l'accesso. Dato che questa vista è in grado di contenere voci per più utenti, include, oltre alle colonne elencate nella Tabella 37.3, una colonna `Owner`. `DBA_TAB_COLUMNS`, che elenca le definizioni delle colonne di tutte le tabelle e le viste del database, ha le stesse definizioni delle colonne di `ALL_TAB_COLUMNS`.

**Tabella 37.3** Colonne di `USER_TAB_COLUMNS`.

IDENTIFICAZIONE	CORRELATE ALLE DEFINIZIONI	CORRELATE ALLE STATISTICHE
Table_Name	Data_Type	Num_Distinct
Column_Name	Data_Length	Low_Value
Column_ID	Data_Precision	High_Value
Character_Set_Name	Data_Scale	Density
Char_Col_Decl_Length	Nullable	Num_Nulls
	Default_Length	Num_Buckets
	Data_Default	Last_Analyzed
	Data_Type_Mod	Sample_Size
	Data_Type_Owner	Global_Stats
	Char_Length	User_Stats
	Char_Used	Avg_Col_Len
	V80_Fmt_Image	
	Data_Upgraded	

## Statistiche sulle colonne

La maggior parte delle statistiche relative alle colonne è disponibile sia in USER\_TAB\_COLUMNS (la loro vecchia collocazione) sia in USER\_TAB\_COL\_STATISTICS. Le colonne disponibili in USER\_TAB\_COL\_STATISTICS sono le colonne “correlate alle statistiche” di USER\_TAB\_COLUMNS, mostrate nella Tabella 37.3, oltre alle colonne Table\_Name e Column\_Name.

USER\_TAB\_COL\_STATISTICS contiene colonne statistiche che sono fornite anche da USER\_TAB\_COLUMNS, per compatibilità con le versioni precedenti. Si dovrebbe accedere a queste colonne tramite USER\_TAB\_COL\_STATISTICS.

## Istogrammi dei valori di colonna

Per migliorare l’analisi utilizzata dall’ottimizzatore basato sui costi, si può ricorrere agli istogrammi. La vista USER\_TAB\_HISTOGRAMS contiene informazioni sull’istogramma di ciascuna colonna, comprese Table\_Name, Column\_Name, Endpoint\_Number, Endpoint\_Value ed Endpoint\_Actual\_Value. I valori di USER\_TAB\_HISTOGRAMS servono all’ottimizzatore per determinare la distribuzione dei valori delle colonne all’interno della tabella. Sono disponibili anche le versioni “ALL” e “DBA” di USER\_TAB\_HISTOGRAMS?

## Colonne aggiornabili

È possibile aggiornare i record di viste che contengono join nelle loro query, purché i join soddisfino alcuni criteri. La vista USER\_UPDATABLE\_COLUMNS elenca tutte le colonne che è possibile aggiornare. È possibile effettuare una query su Owner, Table\_Name e Column\_Name per la colonna; la colonna Updatable avrà il valore YES se può essere aggiornata e il valore NO in caso contrario. Inoltre, è possibile eseguire una query per determinare se sia consentito inserire o eliminare record dalla vista tramite le colonne Insertable e Deletable.

## Viste: USER\_VIEWS

Alla query base di una vista è possibile accedere tramite la vista del dizionario dati USER\_VIEWS, che contiene dieci colonne; le tre colonne principali sono:

View_Name	Nome della vista.
Text_Length	Lunghezza in caratteri della query di base della vista.
Text	Query utilizzata dalla vista.

Le altre colonne, descritte più avanti in questo paragrafo, sono correlate alle viste oggetto.

**NOTA** *Questo paragrafo vale soltanto per le viste tradizionali. Per le viste materializzate, si rimanda al paragrafo “Database link e viste materializzate”, più avanti in questo capitolo.*

La colonna Text è di tipo LONG. Ciò può rappresentare un problema quando si effettua una query su USER\_VIEWS mediante SQLPLUS, poiché SQLPLUS tronca i tipi di dati LONG. Tuttavia, esiste la possibilità di modificare il punto in cui avviene il troncamento con il comando set long. USER\_VIEWS mette a disposizione un meccanismo per stabilire l’impostazione adatta del punto di troncamento dei dati LONG, come descritto nell’esempio seguente.

La colonna Text\_Length visualizza la lunghezza della query della vista. Pertanto, il punto di troncamento dei dati LONG in SQLPLUS dev'essere impostato a un valore uguale o maggiore del valore Text\_Length della vista. Per esempio, nel listato seguente è mostrata una vista il cui nome è AGING e la cui Text\_Length è pari a 355.

```
select View_Name, Text_Length
  from USER_VIEWS
 where View_Name = 'AGING';
```

View_Name	Text_Length
AGING	355

Dato che la lunghezza del testo della vista è pari a 355 caratteri, per incrementare il punto di troncamento dei LONG almeno a 355 (il valore predefinito è 80) occorre il comando **set long**, in modo da visualizzare il testo intero della query della vista:

```
set long 355
```

È possibile eseguire una query su USER\_VIEWS per il testo della vista, con la query riportata nel listato seguente:

```
select Text
  from USER_VIEWS
 where View_Name = 'AGING';
```

Se non si fosse utilizzato il comando **set long**, l'output sarebbe stato troncato a 80 caratteri, senza che alcun messaggio avvertisse del fatto. Prima di effettuare query su altre viste, è necessario controllare nuovamente i rispettivi valori di Text\_Length.

**NOTA** È possibile eseguire la query sulle definizioni di colonna delle viste da USER\_TAB\_COLUMNS, la stessa vista su cui si esegue la query per le tabelle.

Se nelle viste si utilizzano alias per le colonne, e questi fanno parte della query della vista, le query sul dizionario dati per informazioni sulle viste risultano semplificate. Dato che in USER\_VIEWS viene visualizzato il testo interno della query sulla vista, sono visualizzati anche gli alias per le colonne.

Con questo formato si possono creare delle viste:

```
create view RIVISTA_VISTA (AlcuniArticoli, AlcuneSezioni)
  as select Articolo, Sezione
    from RIVISTA;
```

Elencando i nomi delle colonne nell'intestazione del comando **create view** si rimuovono gli alias delle colonne dalle query, evitando così che vengano visualizzati mediante USER\_VIEWS. L'unico modo per visualizzare i nomi delle colonne della vista consisterebbe nell'effettuare una query su USER\_TAB\_COLUMNS. Se i nomi delle colonne si trovano nella query, è sufficiente eseguire una query su una vista del dizionario dati (USER\_VIEWS), sia per la query sia per i nomi delle colonne.

Per esempio, con la vista RIVISTA\_VISTA creata nell'ultimo esempio, eseguendo una query su USER\_VIEWS si visualizzerebbe:

```
select Text
  from USER_VIEWS
 where View_Name = 'RIVISTA_VISTA';

TEXT
-----
select Articolo, Sezione from RIVISTA
```

Questa query *non* mostra i nuovi nomi assegnati alle colonne, in quanto tali nomi non sono stati inseriti nella query sulla vista. Per visualizzare questi nomi di colonna in USER\_VIEWS, è necessario aggiungerli come alias di colonna alla query della vista:

```
create view RIVISTA_VISTA
  as select Articolo AS AlcuniArticoli, Sezione AS AlcuneSezioni
  from RIVISTA;
```

Ora, quando si esegue una query su USER\_VIEWS, gli alias di colonna vengono visualizzati come parte del testo della query sulla vista:

```
select Text
  from USER_VIEWS
 where View_Name = 'RIVISTA_VISTA';

TEXT
-----
select Articolo AlcuniArticoli, Sezione AlcuneSezioni
  from RIVISTA
```

Per supportare le viste dell'oggetto, USER\_VIEWS contiene le colonne seguenti:

Type_Text	Clausola tipo della vista inserita.
Type_Text_Length	Lunghezza della clausola type della vista tipizzata.
OID_Text	Clausola WITH OID della vista tipizzata.
OID_Text_Length	Lunghezza della clausola WITH OID della vista tipizzata.
View_Type_Owner	Owner del tipo della vista per le viste tipizzate.
View_Type	Tipo della vista.

Per ulteriori informazioni su viste oggetto e tipi si rimanda ai Capitoli 30 e 33.

La vista ALL\_VIEWS elenca tutte le viste di proprietà dell'utente e le viste per le quali gli è stato concesso l'accesso. Dato che questa vista è in grado di contenere voci per più utenti, contiene, oltre alle colonne elencate in precedenza, una colonna Owner. DBA\_VIEWS, che elenca tutte le viste del database, ha le stesse definizioni delle colonne di ALL\_VIEWS.

## Sinonimi: USER\_SYNONYMS (SYN)

La vista USER\_SYNONYMS elenca tutti i sinonimi di proprietà dell'utente. Le colonne sono le seguenti:

---

Synonym_Name	Nome del sinonimo.
Table_Owner	Proprietario della tabella cui fa riferimento il sinonimo.
Table_Name	Nome della tabella cui fa riferimento il sinonimo.
DB_Link	Nome del database link utilizzato nel sinonimo.

---

USER\_SYNONYMS è utile per il debug dei programmi o la risoluzione di problemi relativi agli accessi degli utenti a oggetti all'interno di applicazioni. USER\_SYNONYMS è nota anche con il sinonimo pubblico SYN.

Se il sinonimo non utilizza un database link, la colonna DB\_Link ha valore NULL. Pertanto, per visualizzare un elenco dei database link attualmente utilizzati dai sinonimi posseduti dal proprio account è possibile eseguire questa query:

```
select distinct DB_Link
  from USER_SYNONYMS
 where DB_Link is not null;
```

La vista ALL\_SYNONYMS elenca tutti i sinonimi di proprietà dell'utente, i sinonimi pubblici e tutti i sinonimi per i quali all'utente è stato concesso l'accesso. Dato che questa vista è in grado di contenere voci per più utenti, contiene, oltre alle colonne elencate in precedenza, una colonna Owner. DBA\_SYNONYMS, che elenca tutti i sinonimi del database, ha le stesse definizioni delle colonne di ALL\_SYNONYMS.

## Sequenze: USER\_SEQUENCES (SEQ)

Per visualizzare gli attributi delle sequenze, è possibile eseguire una query sulla vista del dizionario dati USER\_SEQUENCES. Su questa vista è possibile eseguire una query anche con il sinonimo pubblico SEQ. Nella Tabella 37.4 sono elencate le colonne della vista USER\_SEQUENCES.

**Tabella 37.4** Colonne della vista USER\_SEQUENCES.

---

NOME COLONNA	DESCRIZIONE
Sequence_Name	Nome della sequenza.
Min_Value	Valore minimo della sequenza.
Max_Value	Valore massimo della sequenza.
Increment_By	Incremento tra due valori della sequenza.
Cycle_Flag	Flag che indica se i valori devono tornare ciclicamente a Min_Value una volta raggiunto Max_Value.
Order_Flag	Flag che indica se i numeri della sequenza sono generati in ordine.
Cache_Size	Numero delle voci di sequenza nella memoria cache.
Last_Number	Ultimo numero di sequenza scritto su disco o nella memoria cache.

---

La colonna Last\_Number non viene aggiornata durante il normale funzionamento del database, ma viene utilizzata durante le operazioni di riavvio e recupero del database stesso.

La vista ALLSEQUENCES elenca tutte le sequenze di proprietà dell'utente o quelle per le quali gli è stato concesso l'accesso. Dato che questa vista è in grado di contenere voci per più utenti, include, oltre alle colonne elencate nella Tabella 37.4, una colonna Sequence\_Owner. DBASEQUENCES, che elenca tutte le sequenze del database, ha le stesse definizioni delle colonne di ALLSEQUENCES.

## 37.4 Vincoli e commenti

I vincoli e i commenti agevolano la comprensione delle relazioni che intercorrono tra tabelle e colonne. I commenti sono strettamente informativi e non impongono alcuna condizione sui dati memorizzati negli oggetti descritti. I vincoli, invece, definiscono le condizioni di validità per i dati. Vincoli tipici sono NOT NULL, UNIQUE, PRIMARY KEY e FOREIGN KEY. Nei paragrafi seguenti, viene descritto come recuperare dati relativi a vincoli e commenti dal dizionario dati.

### Vincoli: USER\_CONSTRAINTS

Le informazioni sui vincoli sono accessibili tramite la vista USER\_CONSTRAINTS. Tali informazioni sono molto utili quando si cerca di modificare i vincoli dei dati o di risolvere problemi con i dati di un'applicazione. Le colonne di questa vista sono elencate nella Tabella 37.5.

Anche se si tratta di una vista “USER”, essa contiene ugualmente una colonna Owner. In questa vista, Owner si riferisce al proprietario del vincolo e non al proprietario della tabella (che è l'utente).

I valori validi per la colonna Constraint\_Type sono descritti nella Tabella 37.5. La comprensione dei tipi di vincoli è fondamentale quando si cerca di ottenere delle informazioni utili sui vincoli stessi.

I vincoli FOREIGN KEY hanno sempre valori per le colonne R\_Owner e R\_Constraint\_Name. Queste due colonne specificano il vincolo cui fa riferimento FOREIGN KEY. FOREIGN KEY fa riferimento a un altro *vincolo* e non a un'altra colonna. I vincoli NOT NULL sulle colonne sono memorizzati come CHECK, pertanto dispongono di un Constraint\_Type uguale a ‘C’.

Con una query su USER\_CONSTRAINTS si ottengono i nomi di tutti i vincoli su una colonna. Ciò è molto importante quando si cerca di interpretare i messaggi di errore che forniscono solamente il nome del vincolo violato.

Non appena si conosce il nome e il tipo di un vincolo, è possibile verificare le colonne a esso associate tramite la vista USER\_CONS\_COLUMNS, descritta nel prossimo paragrafo.

Se non si assegnano ai vincoli dei nomi mentre li si crea, Oracle genererà nomi di vincoli unici. Per ulteriori informazioni, si rimanda al paragrafo “Vincoli di integrità” nel Capitolo 42. Se il nome di un vincolo è stato generato dal sistema, ciò viene indicato nella colonna Generated.

Se si differisce un vincolo (come indicato dalla colonna Deferred), il vincolo in questione non verrà imposto durante la transazione. Per esempio, se si effettuassero aggiornamenti su vasta scala di tabelle correlate e non fosse possibile garantire l'ordine delle transazioni, si potrebbero differire i controlli eseguiti dai vincoli sulle tabelle sino al completamento di tutti gli aggiornamenti.

ALLCONSTRAINTS elenca i vincoli su tutte le tabelle cui l'utente è in grado di accedere. DBA\_CONSTRAINTS elenca tutti i vincoli del database. Ciascuna di queste viste ha lo stesso gruppo di colonne di USER\_CONSTRAINTS (dal momento che contengono tutte la colonna Owner).

**Tabella 37.5** Colonne di USER\_CONSTRAINTS.

NOME COLONNA	DESCRIZIONE
Owner	Proprietario del vincolo.
Constraint_Name	Nome del vincolo.
Constraint_Type	Tipo di vincolo: C: vincolo CHECK; comprende NOT NULL P: vincolo PRIMARY KEY R: vincolo FOREIGN KEY (riferimento) U: vincolo UNIQUE V: vincolo WITH CHECK OPTION (per le viste) O: vincolo WITH READ ONLY (per le viste)
Table_Name	Nome della tabella associata al vincolo.
Search_Condition	Condizione di ricerca utilizzata (per vincoli CHECK).
R_Owner	Proprietario della tabella a cui fa riferimento un vincolo FOREIGN KEY.
R_Constraint_Name	Nome del vincolo a cui fa riferimento un vincolo FOREIGN KEY.
Delete_Rule	Azione da intraprendere sulle tabelle FOREIGN KEY quando viene eliminato un record PRIMARY KEY ('CASCADE' o 'NO ACTION').
Status	Stato del vincolo ('ENABLED' o 'DISABLED').
Deferrable	Flag che indica se il vincolo può essere differito.
Deferred	Flag che indica se il vincolo era inizialmente differito.
Validated	Flag ('VALIDATED' o 'NOT VALIDATED') che indica se tutti i dati obbediscono al vincolo.
Generated	Flag che indica se il nome del vincolo è stato generato dal database.
Bad	Flag che indica se nella creazione del vincolo è stata utilizzata una data senza specificare un valore di secolo per i vincoli CHECK, il che produce anni a due cifre ambigui; si applica solo a vincoli contenuti in database aggiornati da precedenti versioni di ORACLE. Tali vincoli possono causare errori ORA-02436 se non eliminati e ricreati correttamente.
Rely	Flag che indica se il vincolo è imposto o disabilitato.
Last_Change	Ultima data in cui il vincolo è stato abilitato o disabilitato.
Index_Owner	Proprietario dell'indice correlato.
Index_Name	Nome dell'indice correlato.
Invalid	Flag di valid/non valido.
View_Related	Flag Yes/No che registra se il vincolo è correlato alla vista.

### Colonne vincolo: USER\_CONS\_COLUMNS

Le colonne associate ai vincoli possono essere visualizzate tramite la vista del dizionario dati USER\_CONS\_COLUMNS. Se una query su USER\_CONSTRAINTS è già stata eseguita per

ottenere i tipi e i nomi dei vincoli coinvolti, è possibile utilizzare USER\_CONS\_COLUMNS per stabilire quali colonne sono comprese nel vincolo. Le colonne di questa vista sono:

Owner	Proprietario del vincolo.
Constraint_Name	Nome del vincolo.
Table_Name	Nome della tabella associata al vincolo.
Column_Name	Nome della colonna associata al vincolo.
Position	L'ordine delle colonne all'interno della definizione del vincolo.

Sono solo due le colonne contenute in questa vista che non si trovano anche in USER\_CONSTRAINTS: Column\_Name e Position. Nel listato seguente, è riportata una query di esempio su questa tabella:

```
select Column_Name, Position
  from USER_CONS_COLUMNS
 where Constraint_Name = 'SYS_C0008791';

COLUMN_NAME POSITION
----- -----
NOME          1
COGNOME       2
```

Come mostrato nel listato precedente, la combinazione di Nome e Cognome forma il vincolo (in questo caso, una chiave primaria).

La colonna Position è significativa. Quando si crea un vincolo UNIQUE o PRIMARY KEY, Oracle crea automaticamente un indice unico sulla serie di colonne specificate. L'indice viene creato in base all'ordine delle colonne specificato. A sua volta, l'ordine delle colonne influisce sulle prestazioni dell'indice. Un indice che comprende più colonne verrà utilizzato nel modo più efficiente se la sua colonna principale (Position=1) viene utilizzata nella clausola where della query. Per ulteriori informazioni sugli ottimizzatori, si rimanda al Capitolo 38.

Le viste ALL\_CONS\_COLUMNS e DBA\_CONS\_COLUMNS hanno le stesse definizioni delle colonne di USER\_CONS\_COLUMNS. ALL\_CONS\_COLUMNS può essere utilizzata per visualizzare informazioni delle colonne relative ai vincoli su tutte le tabelle cui l'utente può accedere, a prescindere dal proprietario. DBA\_CONS\_COLUMNS elenca le informazioni sui vincoli a livello di colonna per l'intero database.

## Eccezioni ai vincoli: EXCEPTIONS

Quando si attivano vincoli su tabelle che contengono già dei dati è possibile imbattersi in violazioni dei vincoli all'interno dei dati stessi. Per esempio, si potrebbe cercare di creare un vincolo PRIMARY KEY su una colonna che contiene lo stesso valore per più record, tuttavia un tentativo di questo tipo non riuscirebbe a causa delle violazioni ai vincoli di univocità.

Esiste la possibilità di ottenere informazioni sulle righe che causano il fallimento delle creazioni di vincoli. In primo luogo, è necessario creare una tabella di nome EXCEPTIONS nel proprio schema; lo script SQL che si dovrebbe utilizzare per creare questa tabella prende il nome di utlexcpt.sql e normalmente è situato nella directory /rdbms/admin della directory home di Oracle.

La tabella EXCEPTIONS contiene quattro colonne: Row\_ID (il RowID di ciascuna riga che viola il vincolo), Owner (il proprietario del vincolo violato), Table\_Name (la tabella in cui è stato creato il vincolo violato) e Constraint (il vincolo violato dalla riga). Una volta creata la tabella EXCEPTIONS, si può tentare di attivare un vincolo PRIMARY KEY:

```
alter table RIVISTA enable PRIMARY KEY
exceptions into EXCEPTIONS; cannot enable (RIVISTA.SYS_C00516) - primary key violated
```

La creazione del vincolo non è riuscita e nella tabella EXCEPTIONS viene inserito un riferimento a tutte le righe che hanno violato il vincolo. Per esempio, se il vincolo PRIMARY KEY dell'ultimo esempio avesse generato delle eccezioni, sarebbe stato possibile effettuare una query sulla tabella EXCEPTIONS, come mostrato nel listato seguente:

```
select Owner, Table_Name, Constraint from EXCEPTIONS;
```

OWNER	TABLE_NAME	CONSTRAINT
PRATICA	RIVISTA	SYS_C00516
PRATICA	RIVISTA	SYS_C00516

Due righe hanno violato il vincolo chiamato SYS\_C00516 (che, in questo esempio, è il nome assegnato al vincolo PRIMARY KEY per la tabella RIVISTA). È possibile stabilire quali righe della tabella RIVISTA corrispondono a queste eccezioni unendo join la colonna Row\_ID della tabella EXCEPTIONS alla pseudocolonna RowID della tabella in cui è stato collocato il vincolo (in questo esempio, la tabella RIVISTA):

```
select *
  from RIVISTA
 where RowID in
      (select Row_ID from EXCEPTIONS);
```

Vengono così visualizzate le righe che causano la violazione del vincolo.

**NOTA** *Row\_ID è una vera colonna nella tabella EXCEPTIONS, con un tipo di dati ROWID. Viene unita tramite join alla pseudocolonna RowID nella tabella da cui sono state generate le eccezioni.*

## Commenti di tabella: USER\_TAB\_COMMENTS

I commenti possono essere aggiunti a tavole, viste o colonne dopo la loro creazione. I commenti alle viste del dizionario dati sono la base per i record visualizzati mediante le viste DICTIONARY e DICT\_COLUMNS. Per visualizzare i commenti sulle tavole, occorre utilizzare la vista USER\_TAB\_COMMENTS.

La vista USER\_TAB\_COMMENTS contiene tre colonne:

---

Table_Name	Nome della tabella o della vista.
Table_Type	Tipo dell'oggetto (tabella, tabella oggetto o vista).
Comments	Commenti inseriti per l'oggetto.

---

Per aggiungere un commento a una tabella, occorre utilizzare il comando `comment` come nel listato seguente:

```
comment on table COMPLEANNO is 'Date di compleanno degli impiegati di Blacksburg';
```

Si esegua una query su `USER_TAB_COMMENTS` specificando il `Table_Name` per cui si desidera visualizzare i commenti, come mostrato nel listato seguente:

```
select Comments
  from USER_TAB_COMMENTS
 where Table_Name = 'COMPLEANNO';

COMMENTI
-----
Date di compleanno degli impiegati di Blacksburg
```

Per rimuovere un commento, è sufficiente impostare il commento a due apici che non racchiudono alcuno spazio:

```
comment on table COMPLEANNO is '';
```

Per visualizzare i commenti di tutte le tabelle cui è possibile accedere, si utilizza la vista `ALL_TAB_COMMENTS`. Questa vista ha una colonna `Owner` aggiuntiva, che specifica il proprietario della tabella. `DBA_TAB_COMMENTS` elenca tutte le tabelle nel database e possiede anch'essa una colonna `Owner`.

## **Commenti di colonna: `USER_COL_COMMENTS`**

`USER_COL_COMMENTS` visualizza i commenti inseriti per colonne delle tabelle. Questi commenti vengono aggiunti al database tramite il comando `comment`. `USER_COL_COMMENTS` contiene tre colonne:

---

Table_Name	Nome della tabella o della vista.
Column_Name	Nome della colonna.
Comments	Commenti inseriti per la colonna.

---

Si esegua una query su `USER_COL_COMMENTS` specificando il `Table_Name` e il `Column_Name` per i quali si desidera visualizzare i commenti:

```
select Comments
  from USER_COL_COMMENTS
 where Table_Name = 'COMPLEANNO'
   and Column_Name = 'ETA';
```

Per aggiungere un commento a una colonna, si utilizzi il comando `comment` come mostrato nel listato seguente:

```
comment on column COMPLEANNO.ETA is 'Eta in anni';
```

Per rimuovere un commento, è sufficiente impostare il commento a due apici che non racchiudono alcuno spazio:

```
comment on column COMPLEANNO.ETa is '';
```

Per visualizzare i commenti di colonna su tutte le tabelle cui è possibile accedere, si utilizza la vista ALL\_COL\_COMMENTS. Questa vista contiene una colonna Owner aggiuntiva, che specifica il proprietario della tabella. DBA\_COL\_COMMENTS elenca tutte le colonne di tutte le tabelle nel database e possiede anch'essa una colonna Owner.

## 37.5 Indici e cluster

Gli indici e i cluster non modificano i dati memorizzati nelle tabelle, ma solo il modo in cui si memorizzano e si accede a tali dati. Nei paragrafi seguenti vengono introdotte viste del dizionario dati che descrivono indici e cluster. Nel paragrafo “Allocazione e uso dello spazio” di questo capitolo è possibile trovare descrizioni di viste del dizionario dati correlate a indici partizionati.

### Indici: USER\_INDEXES (IND)

In Oracle, gli indici sono strettamente correlati ai vincoli. I vincoli PRIMARY KEY e UNIQUE sono sempre associati a indici unici. Come accade con i vincoli, per eseguire una query sulle informazioni relative agli indici vengono utilizzate due viste del dizionario dati: USER\_INDEXES e USER\_IND\_COLUMNS. USER\_INDEXES è nota anche con il sinonimo pubblico IND.

Le colonne contenute in USER\_INDEXES possono essere suddivise in quattro categorie, come mostrato nella Tabella 37.6.

Il nome dell'indice è riportato nella colonna Index\_Name. Il proprietario e il nome della tabella indicizzata si trovano nelle colonne Table\_Owner e Table\_Name. La colonna Uniqueness è impostata a ‘UNIQUE’ per indici unici e a ‘NONUNIQUE’ per indici non unici. Table\_Type registra se l'indice si trova su una tabella o un cluster.

Le colonne “correlate allo spazio” sono descritte nella voce “Memorizzazione” del Capitolo 42. Le colonne “correlate alle statistiche” sono popolate quando la tabella viene analizzata (si veda il comando analyze nel Capitolo 42). Le colonne Degree e Instances fanno riferimento al grado di parallelismo utilizzato al momento di creare l'indice.

Per visualizzare tutti gli indici di una tabella, occorre effettuare una query su USER\_INDEXES utilizzando le colonne Table\_Owner e Table\_Name nella clausola where, come mostrato nel listato seguente:

```
select Index_Name, Uniqueness
  from USER_INDEXES
 where Table_Owner = 'PRATICA'
   and Table_Name = 'COMPLEANNO';

INDEX_NAME      UNIQUENESS
----- -----
PK_COMPLEANNO    UNIQUE
```

L'output della query nell'esempio precedente dimostra che sulla tabella COMPLEANNO esiste un indice di nome “PK\_COMPLEANNO”. Utilizzando lo stesso standard di denomina-

**Tabella 37.6** Colonne in USER\_INDEX.

IDENTIFICAZIONE	CORRELATE ALLO SPAZIO	CORRELATE ALLE STATISTICHE	ALTRÉ
Index_Name	Tablespace_Name	Blevel	Degree
Table_Owner	Ini_Trans	Leaf_Blocks	Instances
Table_Name	Max_Trans	Distinct_Keys	Include_Column
Table_Type	Initial_Extent	Avg_Leaf_Blocks_Per_Key	Buffer_Pool
Uniqueness	Next_Extent	Avg_Data_Blocks_Per_Key	Duration
Status	Min_Exts	Clustering_Factor	Pct_Direct_Access
Partitioned	Max_Exts	Num_Rows	Parameters
Index_Type	Pct_Increase	Sample_Size	Domidx_Status
Temporary	Pct_Free	Last_Analyzed	Domidx_Opstatus
Generated	Freelists	User_Stats	Funcidx_Status
Logging	Freelist_Groups	Global_Stats	Join_Index
Compression	Pct_Threshold		
Prefix_Length			
Secondary			
Ityp_Owner			
Ityp_Name			

zione per tutti gli indici, è possibile rendere molto facile l'identificazione dello scopo e della tabella cui si riferisce un indice, semplicemente osservando il valore di Index\_Name. In questo esempio, "PK" significa PRIMARY KEY; il nome del vincolo può essere specificato durante la creazione del medesimo per sostituire il nome generato dal sistema.

La colonna Clustering\_Factor non è direttamente correlata ai cluster, ma rappresenta il grado di ordinamento delle righe nella tabella. Più ordinate sono le righe, più efficienti saranno le query d'intervallo, (le *query d'intervallo* sono quelle in cui per una colonna viene dato un intervallo di valori, come in where Cognome like 'A%').

Per scoprire quali colonne fanno parte dell'indice e conoscere il loro ordine all'interno dell'indice stesso, è necessario eseguire una query sulla vista USER\_IND\_COLUMNS, descritta nel prossimo paragrafo.

ALL\_INDEXES visualizza tutti gli indici di proprietà dell'utente e tutti gli indici creati su tabelle cui l'utente ha accesso. Dato che questa vista è in grado di contenere voci per più utenti, include, oltre alle colonne descritte nella Tabella 37.6, una colonna Owner. DBA\_INDEXES, che elenca tutti gli indici del database, ha le stesse definizioni delle colonne di ALL\_INDEXES.

**NOTA**

Per gli indici basati su funzioni, si rimanda a USER\_IND\_EXPRESSIONS.

## Colonne indicizzate: USER\_IND\_COLUMNS

Per determinare quali colonne sono presenti in un indice occorre eseguire una query su USER\_IND\_COLUMNS. Le colonne disponibili tramite questa vista sono:

Index_Name	Nome dell'indice.
Table_Name	Il nome della tabella indicizzata.
Column_Name	Nome di una colonna all'interno dell'indice.
Column_Position	Posizione della colonna all'interno dell'indice.
Column_Length	Lunghezza indicizzata della colonna.
Char_Length	Lunghezza di codepoint massima della colonna (Unicode).
Descend	Un flag Y/N per indicare se la colonna è o meno ordinata in ordine decrescente.

Cinque colonne di questa vista non sono presenti in USER\_INDEXES: Column\_Name, Column\_Position, Column\_Length, Char\_Length e Descend. Column\_Length, come le colonne correlate alle statistiche in USER\_INDEXES, è popolata quando la tabella base dell'indice viene analizzata (si veda il comando `analyze` nel Capitolo 42). Una query di esempio su questa tabella, che utilizza il valore Index\_Name dall'esempio di USER\_INDEXES, è mostrata nel listato seguente (in questo esempio, alla colonna Column\_Position è assegnato l'alias Pos):

```
select Column_Name, Column_Position Pos
  from USER_IND_COLUMNS
 where Index_Name = 'PK_COMPLEANNO';
```

COLUMN_NAME	POS
NOME	1
COGNOME	2

Come si può vedere dalla query, l'indice PK\_COMPLEANNO è costituito da due colonne: nell'ordine, Nome e Cognome. Per dettagli sul modo in cui l'ottimizzatore utilizza gli indici a più colonne, si rimanda al Capitolo 38.

Le viste ALL\_IND\_COLUMNS e DBA\_IND\_COLUMNS hanno le stesse definizioni delle colonne di USER\_IND\_COLUMNS. ALL\_IND\_COLUMNS può essere utilizzata per visualizzare informazioni di colonna relative agli indici su tutte le tabelle cui l'utente può accedere, a prescindere dal proprietario. La vista DBA\_IND\_COLUMNS contiene le informazioni sugli indici a livello di colonna per l'intero database.

## Colonne con indice di join bitmap: USER\_JOIN\_IND\_COLUMNS

Se si crea un indice di join bitmap (caratteristica introdotta in Oracle9i), è possibile eseguire una query su USER\_JOIN\_IND\_COLUMNS per ottenere i dettagli di join. USER\_JOIN\_IND\_COLUMNS registra i nomi delle tabelle coinvolte nel join, le colonne di join (interna ed esterna), nonché la dimensione e le fact table coinvolte.

## Cluster: USER\_CLUSTERS (CLU)

Ai parametri di memorizzazione e statistici associati ai cluster è possibile accedere tramite USER\_CLUSTERS (nota anche con il sinonimo pubblico CLU). Le colonne presenti in questa vista del dizionario dati sono descritte nella Tabella 37.7, suddivise per tipo.

La colonna Cluster\_Name contiene il nome del cluster. La colonna Cluster\_Type specifica se il cluster utilizza un indice B-tree standard o una funzione di hash per il cluster.

L'uso delle colonne "correlate allo spazio" è descritto alla voce "Memorizzazione" nel Capitolo 42. Le colonne "correlate alle statistiche" sono popolate quando la tabella viene analizzata.

ALL\_CLUSTERS e DBA\_CLUSTERS hanno una colonna aggiuntiva, Owner, poiché elencano dei cluster in più schemi.

Per un cluster è possibile specificare una funzione di hash. Le funzioni di hash specificate dall'utente sono particolarmente utili se si verificano le condizioni seguenti:

- La colonna chiave di clusterizzazione è numerica.
- I valori della colonna chiave di clusterizzazione sono assegnati in sequenza.
- Si conosce il numero massimo dei valori della colonna chiave di clusterizzazione per la tabella.

Se tutte e tre le condizioni sono soddisfatte, è possibile utilizzare la clausola hash is del comando create cluster per specificare proprie funzioni di hash (si consulti il paragrafo dedicato a create cluster nel Capitolo 42).

**Tabella 37.7** Colonne di USER\_CLUSTERS.

IDENTIFICAZIONE	CORRELATE ALLO SPAZIO	CORRELATE ALLE STATISTICHE	ALTRE
Cluster_Name	Tablespace_Name	Avg_Blocks_Per_Key	Degree
Cluster_Type	Pct_Free	Hashkeys	Instances
Function	Pct_Used		Cache
	Key_Size		Buffer_Pool
	Ini_Trans		Single_Table
	Max_Trans		Dependencies
	Initial_Extent		
	Next_Extent		
	Min_Exts		
	Max_Exts		
	Pct_Increase		
	Freelists		
	Freelist_Groups		

## Colonne cluster: USER\_CLU\_COLUMNS

Per visualizzare la corrispondenza tra le colonne di una tabella e quelle di un cluster, occorre eseguire una query su USER\_CLU\_COLUMNS, le cui colonne sono:

Cluster_Name	Nome del cluster.
Clu_Column_Name	Nome della colonna chiave del cluster.
Table_Name	Nome di una tabella all'interno del cluster.
Tab_Column_Name	Nome della colonna chiave nella tabella.

Dato che un singolo cluster è in grado di memorizzare dati provenienti da più tabelle, la vista USER\_CLU\_COLUMNS è utile per stabilire quali colonne di quali tabelle corrispondono alle colonne del cluster.

Non esiste una versione “ALL” di questa vista. Esistono solamente USER\_CLU\_COLUMNS per le colonne del cluster dell'utente e DBA\_CLU\_COLUMNS, che visualizza le corrispondenze delle colonne per tutti i cluster del database.

## 37.6 Tipi di dati astratti, strutture correlate a ORDBMS e LOB

Nei prossimi paragrafi verranno introdotte le viste del dizionario dati associate alle strutture relazionali ad oggetti, come tipi di dati astratti, metodi e gli oggetti di grosse dimensioni (LOB). Per tutte queste viste del dizionario dati sono disponibili le versioni “ALL” e “DBA”. Dato che queste viste contengono record per più proprietari, le versioni “ALL” e “DBA” delle viste possiedono, oltre alle colonne elencate in questo paragrafo, anche colonne Owner.

### Tipi di dati astratti USER\_TYPES

I tipi di dati astratti creati nel proprio schema sono elencati in USER\_TYPES, che include delle colonne per Type\_Name, il numero degli attributi (Attributes) e il numero dei metodi (Methods) definiti per il tipo di dati.

Per esempio, il tipo di dati ANIMALE\_TY dispone di tre attributi e un metodo (i metodi sono definiti tramite il comando create type body), come si vede nel listato seguente:

```
select Type_Name,
       Attributes,
       Methods
  from USER_TYPES;
```

TYPE_NAME	ATTRIBUTES	METHODS
ANIMALE_TY	3	1

### Attributi del tipo di dati: USER\_TYPE\_ATTRS

Per visualizzare gli attributi di un tipo di dati, è necessario eseguire una query sulla vista USER\_TYPE\_ATTRS. Le colonne presenti nella vista USER\_TYPE\_ATTRS sono elencate nella Tabella 37.8.

È possibile effettuare una query su USER\_TYPE\_ATTRS per visualizzare le relazioni tra tipi di dati astratti annidati. Per esempio, il tipo di dati PERSONA\_TY utilizza il tipo di dati INDIRIZZO\_TY, come si vede nel listato seguente:

```
select Attr_Name,
       Length,
       Attr_Type_Name
  from USER_TYPE_ATTRS
 where Type_Name = 'PERSONA_TY';
```

ATTR_NAME	LENGTH	ATTR_TYPE_NAME
NOME	25	VARCHAR2
INDIRIZZO		ADDRESS_TY

### Metodi del tipo di dati: USER\_TYPE\_METHODS e USER\_METHOD\_PARAMS

Se per un tipo sono definiti dei metodi, è possibile eseguire una query su USER\_TYPE\_METHODS per stabilire i nomi dei metodi. USER\_TYPE\_METHODS contiene delle colonne in cui sono riportati il nome del tipo (Type\_Name), il nome del metodo (Method\_Name), il numero del metodo (Method\_No, utilizzata per metodi soggetti a overload) e il tipo del metodo (Method\_Type). USER\_TYPE\_METHODS contiene anche colonne in cui sono visualizzati il numero dei parametri (Parameters) e i risultati (Results) restituiti dal metodo.

Per esempio, il tipo di dati ANIMALE\_TY possiede un metodo, una funzione membro di nome ETA. Il metodo ETA possiede un parametro di input (Datanascita) e un output (l'età, in

**Tabella 37.8** Colonne di USER\_TYPE\_ATTRS.

NOME COLONNA	DESCRIZIONE
Type_Name	Nome del tipo.
Attr_Name	Nome dell'attributo.
Attr_Type_Mod	Modificatore del tipo dell'attributo.
Attr_Type_Owner	Proprietario del tipo dell'attributo, se l'attributo stesso è basato su un altro tipo di dati.
Attr_Type_Name	Nome del tipo dell'attributo.
Length	Lunghezza dell'attributo.
Precision	Precisione dell'attributo.
Scale	Scala dell'attributo.
Character_Set_Name	Set di caratteri dell'attributo.
Attr_No	Posizione ordinale dell'attributo all'interno della definizione del tipo.
Inherited	Flag YES/NO che indica se l'attributo è ereditato da un supertipo.

giorni). Una query su USER\_TYPE\_METHODS mostra che per ETA sono definiti *due* parametri, e non uno come ci si aspettava:

```
select Parameters,
       Results
  from USER_TYPE_METHODS
 where Type_Name = 'ANIMALE_TY'
   and Method_Name = 'ETA';

PARAMETERS      RESULTS
-----  -----
2                  1
```

Come mai ETA ha due parametri quando è stato definito un solo parametro di input? Per scoprirlo, è possibile eseguire una query su USER\_METHOD\_PARAMS, che descrive i parametri relativi ai metodi:

```
select Param_Name, Param_No, Param_Type_Name
  from USER_METHOD_PARAMS;

PARAM_NAME          PARAM_NO      PARAM_TYPE_NAME
-----  -----  -----
SELF                1      ANIMALE_TY
DATANASCITA         2      DATE
```

USER\_METHOD\_PARAMS mostra che per ogni metodo Oracle crea un parametro implicito "SELF". I risultati dei metodi sono visualizzati in USER\_METHOD\_RESULTS:

```
select Method_Name,
       Result_Type_Name
  from USER_METHOD_RESULTS
 where Type_Name = 'ANIMALE_TY';

METHOD_NAME          RESULT_TYPE_NAME
-----  -----
ETA                  NUMBER
```

### **Altri tipi di dati: USER\_REFS, USER\_COLL\_TYPES e USER\_NESTED\_TABLES**

Se si usano i REF (si consulti il Capitolo 33), è possibile eseguire una query sulla vista del dizionario dati USER\_REFS per visualizzare i REF definiti. Questa vista mostra il nome della tabella contenente la colonna REF (Table\_Name) e il nome della colonna oggetto (Column\_Name). Anche agli attributi di REF, che specificano se il REF è memorizzato con o senza RowID, è possibile accedere mediante USER\_REFS.

I tipi di collezionatori (tabelle annidate e array variabili) sono descritti attraverso la vista del dizionario dati USER\_COLL\_TYPES. Le colonne di USER\_COLL\_TYPES includono Type\_Name, Upper\_Bound (per gli array variabili), oltre a Length e Precision per gli elementi. La vista USER\_COLL\_TYPES può essere utilizzata insieme alle viste del dizionario dati dei tipi di dati astratti, descritte in precedenza in questo paragrafo, per determinare la struttura del tipo di un collezionatore. Per visualizzare i dettagli delle collezioni, è possibile eseguire una query su USER\_NESTED\_TABLES e USER\_VARRAYS.

## LOB: USER\_LOBS

Come descritto nel Capitolo 32, all'interno del database è possibile memorizzare oggetti di grandi dimensioni (LOB, Large Objects). La vista USER\_LOBS fornisce informazioni sui LOB definiti nelle tabelle, come mostrato nel listato seguente:

```
column Column_Name format A30
select Table_Name,
       Column_Name
  from USER_LOBS;
```

TABLE_NAME	COLUMN_NAME
PROPOSTA	TESTO_PROPOSTA
PROPOSTA	BUDGET

USER\_LOBS visualizza anche i nomi dei segmenti utilizzati per contenere i dati LOB quando questi aumentano; essa tuttavia non visualizza i tipi di dati delle colonne LOB. Per visualizzare i tipi di dati LOB, è possibile descrivere la tabella che contiene quei LOB oppure eseguire una query su USER\_TAB\_COLUMNS (come mostrato in precedenza).

Per utilizzare i tipi di dati BFILE per i LOB, è necessario creare delle directory (si rimanda alla voce del comando `create directory` nel Capitolo 42). Il dizionario dati ALL\_DIRECTORIES visualizza i valori di Owner, Directory\_Name e Directory\_Path per ogni directory cui è stato concesso l'accesso. Per questa vista è disponibile anche una versione “DBA”, mentre non esiste la versione “USER”.

## 37.7 Database link e viste materializzate

I database link e le viste materializzate servono per gestire l'accesso ai dati remoti. In base al tipo di vista materializzata impiegata, è possibile utilizzare i log di viste materializzate. Nei paragrafi seguenti sono descritte viste del dizionario dati che possono essere utilizzate per visualizzare informazioni sui database link e sulle viste materializzate. Per ulteriori informazioni sui database link si rimanda al Capitolo 22. Per ulteriori informazioni sulle viste materializzate si rimanda al Capitolo 23.

### Database link: USER\_DB\_LINKS

Per visualizzare i database link creati nel proprio account, occorre effettuare una query su USER\_DB\_LINKS. Le colonne di questa vista, compresi il nome del link (DB\_Link), il nome utente cui connettersi (Username) e la stringa di connessione (Host) mostrano le informazioni relative alla connessione remota che il link è solito stabilire. I valori di Username e Password sono utilizzati per connettersi al database remoto definito dal valore di Host.

La colonna Host memorizza i nomi di servizio di rete Oracle. Questa colonna memorizza l'esatta stringa di caratteri specificata durante l'esecuzione del comando `create database link` e ne non altera le maiuscole e le minuscole. Di conseguenza, occorre prestare particolare attenzione alle minuscole e maiuscole utilizzate quando si creano i database link, altrimenti le query eseguite su USER\_DB\_LINKS dovranno prendere in considerazione incoerenze di lettere mai-

scole e minuscole nella colonna Host. Per esempio, la ricerca di un database link che utilizza il descrittore di servizio 'HQ' richiede l'inserimento di:

```
select * from USER_DB_LINKS
where UPPER(Host) = 'HQ';
```

dal momento che è possibile che esistano voci con valore di Host uguale a 'hq' invece di 'HQ'.

**NOTA** *Se si utilizzano connessioni definite al database remoto, Password sarà NULL.*

La vista ALL\_DB\_LINKS elenca tutti i database link che sono di proprietà dell'utente o PUBLIC. DBA\_DB\_LINKS elenca tutti i link nel database. Moltissime definizioni delle colonne di ALL\_DB\_LINKS e DBA\_DB\_LINKS sono uguali a quelle di USER\_DB\_LINKS; queste definizioni hanno però una colonna Owner al posto della colonna Password.

Per ulteriori informazioni sull'impiego dei database link, si rimanda al Capitolo 22.

## Viste materializzate

Per visualizzare informazioni relative alle viste materializzate possedute dal proprio account, è possibile eseguire una query su USER\_MVIEWS. Questa vista, le cui colonne sono elencate nella Tabella 37.9, mostra le informazioni strutturali sulla vista materializzata completa dei relativi intervalli di aggiornamento.

**NOTA** *Per informazioni dettagliate sulle viste materializzate, si rimanda al Capitolo 23.*

Il nome della vista materializzata è riportato nella colonna Mview\_Name di USER\_MVIEWS. La tabella locale di base per questa vista è la colonna Container\_Name.

Per determinare quali database link sono utilizzati dalle varie viste materializzate, è necessario effettuare una query sulla colonna Master\_Link, come mostrato nell'esempio seguente:

```
select Master_Link
from USER_MVIEWS;
```

I nomi dei database link restituiti da questa query possono essere utilizzati come input per query eseguite su USER\_DB\_LINKS. Questa query visualizza tutte le informazioni disponibili sui database link utilizzati nelle viste materializzate:

```
select *
from USER_DB_LINKS
where DB_Link in
(select Master_Link
from USER_MVIEWS);
```

Nel Capitolo 23 sono descritte altre query utili per la gestione delle viste materializzate.

Le viste ALL\_MVIEWS e DBA\_MVIEWS hanno le stesse definizioni delle colonne di USER\_MVIEWS. ALL\_MVIEWS può essere utilizzata per visualizzare informazioni su tutte le viste materializzate cui l'utente può accedere, a prescindere dal proprietario. DBA\_MVIEWS elenca informazioni sulle viste materializzate per tutti gli utenti del database.

**Tabella 37.9** Colonne USER\_MVIEWS.

NOME COLONNA	DESCRIZIONE
Owner	L'account proprietario della vista materializzata.
Mview_Name	Nome della vista materializzata.
Container_Name	La tabella di base (nel database locale) per i dati della vista materializzata; per i db precedenti a Oracle8i, corrisponde a Mview_Name preceduto da "SNAP\$".
Query	La query che definisce la vista materializzata.
Query_Len	Lunghezza della query di base della vista materializzata.
Updatable	Flag che indica se la vista materializzata può essere aggiornata.
Update_Log	Nome della tabella che registra le modifiche apportate a una vista materializzata aggiornabile.
Master_Rollback_Seg	Il segmento di rollback da usare durante la popolazione della vista materializzata e le operazioni di aggiornamento.
Master_Link	Il database link usato per accedere al database master.
Rewrite_Enabled	Flag Y/N che indica se la riscrittura della query è abilitata per la vista.
Rewrite_Capability	Regole e restrizioni per le riscritture delle query.
Refresh_Mode	DEMAND, COMMIT o NEVER, in relazione alla modalità di aggiornamento per la vista.
Refresh_Method	Valori usati per guidare un aggiornamento rapido della vista (Complete, Fast, Never o Force).
Build_Mode	Istanziamento IMMEDIATE, DEFERRED o PREBUILT dei dati della vista materializzata durante la sua creazione.
Fast_Refresable	Metodi disponibili per un aggiornamento rapido della vista materializzata.
Last_Refresh_Type	Il tipo di aggiornamento più recente utilizzato.
Last_Refresh_Date	Indicatore orario che registra l'ora dell'ultimo aggiornamento dei dati della vista materializzata.
Staleness	Stato dei dati della vista materializzata in relazione alle loro tabelle master.
After_Fast_Refresh	Stato di validità che segue un aggiornamento rapido.
Compile_State	Validità della vista materializzata.
Use_No_Index	Flag Y/N che indica se la vista materializzata è stata creata con la clausola USING NO INDEX.

Due viste correlate, USER\_REFRESH e USER\_REFRESH\_CHILDREN, visualizzano informazioni sui gruppi di aggiornamento. USER\_MVIEW\_REFRESH\_TIMES mostra la data dell'ultimo aggiornamento per ogni vista materializzata.

### Altre capacità delle viste materializzate

Per visualizzare le viste materializzate che supportano la riscrittura delle query, è possibile eseguire una query su USER\_MVIEW\_ANALYSIS. Se una vista materializzata è stata creata come snapshot prima di Oracle8i, oppure se contiene dei riferimenti a una tabella remota, allora non

verrà elencata in questa vista. È possibile eseguire una query sul nome del proprietario della vista materializzata, sul nome della vista (Mview\_Name) e sul nome del proprietario della tabella base (Mview\_Table\_Owner). Molte colonne di questa vista sono dei flag, come Summary (Y se la vista contiene un'aggregazione), Known\_Stale (Y se i dati della vista sono incoerenti rispetto alla tabella base) e Contains\_VIEWS (Y se la vista materializzata fa riferimento a una vista).

Se la vista materializzata contiene delle aggregazioni, è possibile eseguire una query su USER\_MVIEW\_AGGREGATES per ottenere i dettagli relativi alle aggregazioni. Le colonne di questa vista sono:

Owner	Proprietario della vista materializzata.
Mview_Name	Nome della vista materializzata.
Position_in_Select	Posizione all'interno della query.
Container_Column	Nome della colonna.
Agg_Function	Funzione di aggregazione.
DistinctFlag	'Y' se l'aggregazione utilizza la funzione DISTINCT.
Measure	Testo SQL della misura, che esclude però la funzione di aggregazione.

È possibile eseguire una query sui dettagli delle relazioni all'interno delle viste materializzate partendo dalle viste del dizionario dati USER\_MVIEW\_DETAIL\_RELATIONS e USER\_MVIEW\_KEYS. Se la vista materializzata si basa sui join, per i dettagli sui join si analizzi USER\_MVIEW\_JOINS. In generale, USER\_MVIEW\_ANALYSIS è la vista del dizionario dati più utilizzata per quanto riguarda le viste materializzate.

## Log delle viste materializzate: USER\_MVIEW\_LOGS

I log delle viste materializzate possono essere utilizzati da molte viste materializzate per stabilire quali record della tabella principale devono essere aggiornati nella vista materializzata di questa tabella. Si può eseguire una query su USER\_MVIEW\_LOGS per ottenere informazioni sui log di un utente, tra cui il nome della tabella principale (Master), il nome della tabella che contiene i record di log (Log\_Table) e se la vista materializzata si basa sulla chiave primaria o sul valore di RowID (le colonne Primary\_Key e Rowids). In genere le query su USER\_MVIEWS\_LOGS vengono effettuate a scopo gestionale, per esempio per stabilire il nome del trigger utilizzato per creare i record di log delle viste materializzate.

Non esiste una versione "ALL" di questa vista. DBA\_MVIEWS\_LOGS ha le stesse definizioni delle colonne di USER\_MVIEWS\_LOGS, e visualizza tutti i log delle viste materializzate nel database. È possibile eseguire una query su USER\_BASE\_TABLE\_MVIEWS per ottenere un elenco delle tabelle principali delle viste materializzate che usano i log delle viste materializzate.

## 37.8 Trigger, procedure, funzioni e package

Le procedure, i package e i trigger, blocchi di codice PL/SQL memorizzati nel database, possono essere utilizzati per imporre regole business o eseguire elaborazioni complesse. I trigger sono

descritti nel Capitolo 28, mentre le procedure, le funzioni e i package sono l'argomento del Capitolo 29. Nei paragrafi seguenti sono spiegate le query sul dizionario dati necessarie per ottenere informazioni su trigger, procedure, package e funzioni.

## Trigger: USER\_TRIGGER

USER\_TRIGGER contiene informazioni riguardanti i trigger posseduti dal proprio account. Questa vista, le cui colonne sono elencate nella Tabella 37.10, visualizza il tipo e il corpo del trigger.

La vista ALL\_TRIGGER elenca i trigger per tutte le tabelle cui l'utente ha accesso. DBA\_TRIGGER elenca tutti i trigger nel database. Entrambe le viste contengono una colonna aggiuntiva, Owner, che registra il proprietario del trigger.

Una seconda vista del dizionario dati relativa ai trigger, USER\_TRIGGER\_COLS, mostra in che modo un trigger utilizza le colonne. La vista elenca il nome di ciascuna colonna su cui agisce un trigger, e l'uso del trigger stesso. Come per USER\_TRIGGER, anche per questa vista del dizionario dati sono disponibili le versioni "ALL" e "DBA".

## Procedure, funzioni e package: USER\_SOURCE

Il codice sorgente per procedure, funzioni, package e corpi di package già esistenti può essere sottoposto a una query dalla vista del dizionario dati USER\_SOURCE. La colonna Type di USER\_SOURCE identifica l'oggetto procedurale come 'PROCEDURE', 'FUNCTION',

**Tabella 37.10** Colonne di USER\_TRIGGER.

NOME COLONNA	DESCRIZIONE
Trigger_Name	Nome del trigger.
Trigger_Type	Tipo del trigger ('BEFORE STATEMENT', 'BEFORE EACH ROW' e così via).
Triggering_Event	Comando che esegue il trigger ('INSERT', 'UPDATE' o 'DELETE').
Table_Owner	Proprietario della tabella per cui è definito il trigger.
Base_Object_Type	Tipo dell'oggetto sul quale si basa il trigger (TABLE, VIEW, SCHEMA o DATABASE).
Table_Name	Nome della tabella per cui è definito il trigger.
Column_Name	Per i trigger delle tabelle annidate, il nome della colonna che contiene la tabella annidata.
Referencing_Names	Nomi utilizzati per fare riferimento a valori OLD e NEW nel trigger.
When_Clause	Clausola when utilizzata per il trigger.
Status	Stato di ENABLED o DISABLED del trigger.
Description	Descrizione del trigger.
Action_Type	Tipo di azione per il corpo del trigger (CALL o PL/SQL).
Trigger_Body	Testo del trigger.

‘PACKAGE’, ‘PACKAGE BODY’, ‘TRIGGER’, ‘TYPE’, ‘TYPE BODY’, o ‘JAVA SOURCE’. Ciascuna linea di codice viene memorizzata in un record separato di USER\_SOURCE.

Si possono selezionare delle informazioni dalla vista USER\_SOURCE con una query simile a quella riportata nel listato seguente. In questo esempio, la colonna Text viene selezionata e ordinata per numero di linea. Il nome dell’oggetto e il suo tipo sono utilizzati per specificare gli oggetti per i quali è necessario visualizzare il codice sorgente.

```
select Text
  from USER_SOURCE
 where Name = '&procedure_name'
   and Type = 'PROCEDURE'
 order by Line;
```

**NOTA** *La sequenza delle linee viene mantenuta dalla colonna Line. Per questo motivo, la colonna Line dovrebbe essere utilizzata nella clausola order by.*

Le viste ALL\_SOURCE e DBA\_SOURCE contengono tutte le colonne presenti in USER\_SOURCE, più una colonna aggiuntiva Owner (il proprietario dell’oggetto). ALL\_SOURCE può essere utilizzata per visualizzare il codice sorgente di tutti gli oggetti procedurali cui l’utente può accedere, indipendentemente dal proprietario. DBA\_SOURCE elenca il codice sorgente per tutti gli utenti del database.

**NOTA** *Per visualizzare le impostazioni di parametri persistenti per le unità PL/SQL, occorre eseguire una query su USER\_STORED\_SETTINGS.*

### >Errori di codice: USER\_ERRORS

Il comando SQLPLUS show errors verifica la vista del dizionario dati USER\_ERRORS alla ricerca degli errori associati al tentativo di compilazione più recente di un oggetto procedurale. Il comando show errors visualizza il numero di linea e colonna per ciascun errore, insieme al testo del messaggio relativo.

Per visualizzare gli errori associati a oggetti procedurali creati in precedenza, è possibile effettuare una query diretta su USER\_ERRORS. Ciò può rendersi necessario quando si visualizzano gli errori associati ai corpi di package, in quanto la compilazione di un package che restituisce un errore può non visualizzare l’errore nel corpo di package quando si esegue il comando show error. Una query su USER\_ERRORS può anche rendersi necessaria quando si verificano errori di compilazione con più oggetti procedurali.

Di seguito sono elencate le colonne disponibili in USER\_ERRORS:

---

Name	Nome dell’oggetto procedurale.
Type	Tipo di oggetto (‘PROCEDURE’, ‘FUNCTION’, ‘PACKAGE’, ‘PACKAGE BODY’, ‘TRIGGER’, ‘TYPE’, ‘TYPE BODY’, o ‘JAVA SOURCE’).
Sequence	Numero di sequenza della linea, da usare nella clausola <i>order by</i> della query.
Line	Numero di linea all’interno del codice sorgente in cui si è verificato l’errore.
Position	Posizione all’interno della linea in cui è avvenuto l’errore.
Text	Testo del messaggio di errore.

---

Le query eseguite su questa vista dovrebbero includere sempre la colonna Sequence nella clausola order by. Per questa vista sono disponibili anche le versioni “ALL” e “DBA”; esse possiedono una colonna in più, Owner, che registra il proprietario dell’oggetto.

### **Dimensione del codice: USER\_OBJECT\_SIZE**

La quantità di spazio utilizzato nella tablespace SYSTEM per un oggetto procedurale può essere sottoposta a una query dalla vista del dizionario dati USER\_OBJECT\_SIZE. Come illustra il listato seguente, le quattro aree dimensionali separate possono essere sommate per determinare lo spazio totale utilizzato nelle tabelle del dizionario dati SYSTEM per memorizzare l’oggetto. Le quattro colonne Size, insieme alle colonne Name e Type, rappresentano tutte le colonne presenti in questa vista.

```
select Source_Size+Code_Size+Parsed_Size+Error_Size Total
  from USER_OBJECT_SIZE
 where Name = '&procedure_name'
   and Type = 'PROCEDURE';
```

Per questa vista esiste anche la versione “DBA”. DBA\_OBJECT\_SIZE elenca le dimensioni di tutti gli oggetti contenuti nel database.

## **37.9 Dimensioni**

Esiste la possibilità di creare e gestire dimensioni e gerarchie. Per esempio, si potrebbe avere una tabella di nome PAESE, con delle colonne denominate Paese e Continente. Una seconda tabella, di nome CONTINENTE, potrebbe avere una colonna chiamata Continente. Se si dispone del privilegio di sistema CREATE DIMENSION, è possibile creare una dimensione che riflette la relazione esistente tra questi elementi:

```
create dimension GEOGRAFIA
  level PAESE_ID      is PAESE.Paese
  level CONTINENTE_ID is CONTINENTE.Continente
  hierarchy PAESE_ROLLUP (
    PAESE_ID           child of
    CONTINENTE_ID
  join key PAESE.Continente references CONTINENTE_id);
```

Per visualizzare i nomi delle dimensioni, è possibile eseguire una query sulla colonna Dimension\_Name della vista USER\_DIMENSIONS. USER\_DIMENSIONS contiene anche colonne per il proprietario della dimensione (Owner), per lo stato (la colonna Invalid, impostata a “Y” o “N”) e per il livello di revisione (la colonna Revision). Tramite altre viste del dizionario dati è possibile accedere agli attributi di una dimensione.

Per visualizzare le gerarchie all’interno di una dimensione, è necessario eseguire una query su USER\_DIM\_HIERARCHIES. Questa vista possiede solo tre colonne: Owner, Dimension\_Name e Hierarchy\_Name. L’esecuzione di una query su USER\_DIM\_HIERARCHIES per la dimensione GEOGRAFIA restituisce il nome delle sue gerarchie:

```
select Hierarchy_Name
  from USER_DIM_HIERARCHIES;
-----  
HIERARCHY_NAME  
-----  
PAESE_ROLLUP
```

Per visualizzare i dettagli della gerarchia per PAESE\_ROLLUP è sufficiente eseguire una query su USER\_DIM\_CHILD\_OF, come mostrato nel listato seguente:

```
column chiave_join_id format a4
```

```
select Child_Level_Name,
       Parent_Level_Name,
       Position, Join_Key_Id
  from USER_DIM_CHILD_OF
 where Hierarchy_Name= 'PAESE_ROLLUP';
```

CHILD_LEVEL_NAME	PARENT_LEVEL_NAME	POSITION JOIN
PAESE_ID	CONTINENTE_ID	1 1

Per visualizzare la chiave di join di una gerarchia, è necessario eseguire una query su USER\_DIM\_JOIN\_KEY:

```
select Level_name, Child_Join_Column
  from USER_DIM_JOIN_KEY
 where Dimension_Name= 'GEOGRAFIA'
   and Hierarchy_Name= 'PAESE_ROLLUP';
```

LEVEL_NAME	CHILD_JOIN_COLUMN
CONTINENTE_ID	CONTINENTE

I livelli di una dimensione sono visibili se si esegue una query sulla vista del dizionario dati USER\_DIM\_LEVELS, mentre le colonne di chiave per i vari livelli sono visibili tramite USER\_DIM\_LEVEL\_KEY. Tramite USER\_DIM\_ATTRIBUTES è possibile accedere alle informazioni sugli attributi delle dimensioni.

Per tutte le viste del dizionario dati relative alle dimensioni esistono le viste "ALL" e "DBA". Dato che le viste "USER" per le dimensioni contengono una colonna Owner, nelle viste "ALL" e "DBA" corrispondenti non si trovano colonne aggiuntive.

### **37.10 Allocazione e uso dello spazio, comprese partizioni e partizioni secondarie**

È possibile effettuare una query sul dizionario dati per stabilire lo spazio disponibile e allocato per gli oggetti del database. Nei paragrafi seguenti, viene spiegato come definire i parametri di memorizzazione predefiniti per gli oggetti, la propria quota di utilizzo dello spazio, lo spazio libero disponibile e il modo in cui gli oggetti vengono fisicamente memorizzati. Per informazioni sui metodi di memorizzazione dei dati applicati da Oracle, si rimanda ai Capitoli 20 e 40.

#### **Tablespace: USER\_TABLESPACES**

È possibile eseguire una query sulla vista del dizionario dati USER\_TABLESPACES per stabilire a quali tablespace è stato concesso l'accesso e i parametri di memorizzazione predefiniti di ciascuna di esse. Questi parametri vengono utilizzati per ciascun oggetto memorizzato all'interno di questa tablespace, a meno che un comando create o alter specifichi per tale oggetto dei parametri di memorizzazione differenti. Le colonne correlate alla memorizzazione della vista

USER\_TABLESPACES, elencate nella Tabella 37.11, sono molto simili alle colonne correlate alla memorizzazione di USER\_TABLES. Per ulteriori informazioni si rimanda alla voce “Memorizzazione” del Capitolo 42.

Non esiste una versione “ALL” di questa vista. DBA\_TABLESPACES visualizza i parametri di memorizzazione per tutte le tablespaces.

### Quote di spazio: USER\_TS\_QUOTAS

La vista USER\_TS\_QUOTAS è molto utile per determinare lo spazio attualmente allocato per l’utente e la quantità massima di spazio disponibile per tablespace. Nel listato seguente è riportata una query di esempio su USER\_TS\_QUOTAS:

```
select * from USER_TS_QUOTAS;
```

TABLESPACE_NAME	BYTES	MAX_BYTES	BLOCKS	MAX_BLOCKS
USERS	67584	0	33	0

USER\_TS\_QUOTAS contiene un record per ciascun valore di Tablespace\_Name. La colonna Bytes riporta il numero di byte allocati per gli oggetti di proprietà dell’utente. Max\_Bytes è il

**Tabella 37.11** Colonne di USER\_TABLESPACES.

NOME COLONNA	DESCRIZIONE
Tablespace_Name	Nome della tablespace.
Block_Size	Dimensione del blocco di database in uso per questa tablespace.
Initial_Extent	Parametro INITIAL predefinito per gli oggetti della tablespace.
Next_Extent	Parametro NEXT predefinito per gli oggetti della tablespace.
Min_Exts	Parametro MINEXTENTS predefinito per gli oggetti della tablespace.
Max_Exts	Parametro MAXEXTENTS predefinito per gli oggetti della tablespace.
Pct_Increase	Parametro PCTINCREASE predefinito per gli oggetti della tablespace.
Min_Extlen	Dimensioni predefinite dell’estensione minima per gli oggetti della tablespace.
Status	Stato della tablespace (‘ONLINE’, ‘OFFLINE’, ‘INVALID’, ‘READ ONLY’). Una tablespace “invalid” è una tablespace eliminata. In questa vista i suoi record sono ancora visibili.
Contents	Flag che indica se la tablespace viene utilizzata per memorizzare oggetti permanenti (‘PERMANENT’) o solo segmenti temporanei (‘TEMPORARY’).
Logging	Flag che indica il valore per il parametro LOGGING/NOLOGGING predefinito per gli oggetti della tablespace.
Extent_Management	Luogo dove la gestione degli extent viene effettuata per la tablespace (‘DICTIONARY’ o ‘LOCAL’).
Allocation_Type	Tipo di allocazione degli extent in uso.
Segment_Space_Management	Flag che indica se lo spazio libero viene gestito tramite free list (MANUAL) o bitmap (AUTO).

numero massimo di byte che l'utente può possedere in questa tablespace; se per questa tablespace non ci sono quote disponibili, allora Max\_Bytes visualizzerà un valore pari a 0. Le colonne Bytes e Max\_Bytes vengono convertite in blocchi Oracle, rispettivamente nelle colonne Blocks e Max\_Blocks.

Non esiste una versione “ALL” di questa vista. DBA\_TS\_QUOTAS visualizza le quote di memorizzazione per tutti gli utenti e per tutte le tablespace, e rappresenta un modo molto efficace per elencare l'uso dello spazio in tutto il database.

## **Segmenti ed extent: USER\_SEGMENTS e USER\_EXTENTS**

Come descritto nel Capitolo 20, lo spazio viene allocato per oggetti (come tabelle, cluster e indici) in *segmenti*, ovvero le controparti fisiche degli oggetti logici creati nel database. Per visualizzare i parametri di memorizzazione correnti e lo spazio effettivamente utilizzato per i segmenti, è possibile effettuare una query su USER\_SEGMENTS. Questa vista è molto utile quando esiste il pericolo di superare uno dei limiti di memorizzazione. Le colonne presenti in questa vista sono elencate nella Tabella 37.12.

**Tabella 37.12** Colonne di USER\_SEGMENTS.

NOME COLONNA	DESCRIZIONE
Segment_Name	Nome del segmento.
Partition_Name	NULL se l'oggetto non è partizionato. Altrimenti, il nome della partizione del segmento.
Segment_Type	Tipo del segmento ('TABLE', 'CLUSTER', 'INDEX', 'ROLLBACK' e così via).
Tablespace_Name	Nome della tablespace in cui il segmento viene memorizzato.
Bytes	Numero di byte allocati al segmento.
Blocks	Numero di blocchi ORACLE allocati al segmento.
Extents	Numero di estensioni nel segmento.
Initial_Extent	Dimensioni dell'extent iniziale del segmento.
Next_Extent	Valore del parametro NEXT per il segmento.
Min_Exents	Numero minimo di extent nel segmento.
Max_Exents	Valore del parametro MAXEXTENTS per il segmento.
Pct_Increase	Valore del parametro PCTINCREASE per il segmento.
Freelists	Numero di “freelist” (elenchi di blocchi di dati nel segmento che possono essere utilizzati durante gli inserimenti) allocati al segmento. Se un segmento prevede più freelist, la contesa per i blocchi liberi durante gli insert contemporanei si riduce.
Freelist_Groups	Numero di gruppi di freelist (da utilizzare con l'opzione Parallel Server) allocati al segmento.
Buffer_Pool	Buffer pool nel quale il segmento verrà letto ('DEFAULT', 'KEEP' o 'RECYCLE') se sono stati definiti più buffer pool.

I segmenti sono costituiti da sezioni attigue, chiamate *extent*. Gli extent che costituiscono i segmenti sono descritti in USER\_EXTENTS. Nella vista USER\_EXTENTS è possibile visualizzare la dimensione effettiva di ciascun extent all'interno del segmento; ciò si rivela molto utile per tenere traccia dell'impatto delle modifiche apportate alle impostazioni next e pctincrease. Oltre alle colonne Segment\_Name, Segment\_Type e Tablespace\_Name, USER\_EXTENTS contiene quattro nuove colonne: Extent\_ID (per identificare l'extent all'interno del segmento), Bytes (la dimensione dell'extent, in byte), Blocks (la dimensione dell'extent, in blocchi di Oracle) e Partition\_Name (se il segmento fa parte di un oggetto partizionato).

Sia USER\_SEGMENTS, sia USER\_EXTENTS dispongono di versioni “DBA”, utili per elencare l'impiego di spazio degli oggetti tra i vari proprietari. Sia DBA\_SEGMENTS, sia DBA\_EXTENTS prevedono una colonna in più, Owner. Se si desidera elencare tutti i proprietari che possiedono segmenti di una tablespace, è possibile effettuare una query basata sulla colonna Tablespace\_Name in DBA\_SEGMENTS ed elencare tutti i proprietari di segmenti in questa tablespace.

## Partizioni e partizioni secondarie

I dati di una singola tabella possono essere memorizzati in più partizioni. Per esempi di partizioni e una descrizione delle opzioni di indicizzazione disponibili, si rimanda al Capitolo 20. Per vedere in che modo è stata partizionata una tabella, occorre eseguire una query sulla vista del dizionario dati USER\_PART\_TABLES, le cui colonne sono elencate per categoria nella Tabella 37.13.

La maggior parte delle colonne di USER\_PART\_TABLES definisce i parametri di memorizzazione predefiniti per le partizioni della tabella. Quando alla tabella si aggiunge una partizione, questa utilizza per default i parametri di memorizzazione contenuti in USER\_PART\_TABLES.

**Tabella 37.13** Colonne di USER\_PART\_TABLES.

IDENTIFICAZIONE	CORRELATE ALLA MEMORIZZAZIONE
Table_Name	Def_Tablespace_Name
Partitioning_Type	Def_Pct_Free
Subpartitioning_Type	Def_Pct_Used
Partition_Count	Def_Ini_Trans
Def_Subpartition_Count	Def_Max_Trans
Partitioning_Key_Count	Def_Initial_Extent
Subpartitioning_Key_Count	Def_Next_Extent
DefLogging	Def_Min_Exts
Def_Buffer_Pool	Def_Max_Exts
	Def_Pct_Increase
	Def_Freelists
	Def_Freelist_Groups

La vista `USER_PART_TABLES` visualizza anche il numero di partizioni della tabella (`Partition_Count`), il numero di colonne nella chiave di partizione (`Partitioning_Key_Count`) e il tipo di partizionamento (`Partitioning_Type`).

`USER_PART_TABLES` memorizza una singola riga per ciascuna tabella partizionata. Per visualizzare informazioni su ciascuna delle singole partizioni appartenenti alla tabella, è possibile effettuare una query su `USER_TAB_PARTITIONS`. In `USER_TAB_PARTITIONS` verrà visualizzata una riga per ciascuna partizione della tabella. Le colonne della vista `USER_TAB_PARTITIONS` sono elencate, per categoria, nella Tabella 37.14.

`USER_TAB_PARTITIONS` contiene colonne che identificano la tabella cui appartiene la partizione e visualizza i parametri di memorizzazione della partizione e le statistiche per la partizione. Le colonne “correlate alle statistiche” sono popolate quando la tabella viene analizzata (si veda il comando `analyze` nel Capitolo 42). Le colonne “identificazione” mostrano il valore superiore per l’intervallo utilizzato per definire la partizione (`High_Value`) e la posizione della partizione all’interno della tabella (`Partition_Position`).

Alle colonne impiegate per la chiave di partizione è possibile accedere tramite la vista del dizionario dati `USER_PART_KEY_COLUMNS`. `USER_PART_KEY_COLUMNS` contiene solo quattro colonne:

Name	Nome della tabella o indice partizionati.
Object_Type	Tipo di oggetto (TABELLA o INDICE).
Column_Name	Nome della colonna che fa parte della chiave di partizione.
Column_Position	Posizione della colonna all’interno della chiave di partizione.

**Tabella 37.14** Colonne di `USER_TAB_PARTITIONS`.

IDENTIFICAZIONE	CORRELATE ALLA MEMORIZZAZIONE	CORRELATE ALLE STATISTICHE
Table_Name	Tablespace_Name	Num_Rows
Composite	Pct_Free	Blocks
Partition_Name	Pct_Used	Empty_Blocks
Subpartition_Count	Ini_Trans	Avg_Space
High_Value	Max_Trans	Chain_Cnt
High_Value_Length	Initial_Extent	Avg_Row_Len
Partition_Position	Next_Extent	Sample_Size
Logging	Min_Extent	Last_Analyzed
Buffer_Pool	Max_Extent	Global_Stats
	Pct_Increase	User_Stats
	Freelists	
	Freelist_Groups	

Alle statistiche per le colonne di partizione è possibile accedere tramite la vista del dizionario dati USER\_PART\_COL\_STATISTICS. Le colonne di USER\_PART\_COL\_STATISTICS assomigliano molto a quelle di USER\_TAB\_COL\_STATISTICS.

La distribuzione dei dati all'interno delle partizioni viene registrata durante l'esecuzione del comando `analyze`. Alle informazioni sull'istogramma dei dati per le partizioni è possibile accedere tramite la vista del dizionario dati USER\_PART\_HISTOGRAMS. Le colonne di questa vista sono Table\_Name, Partition\_Name, Column\_Name, Bucket\_Number, Endpoint\_Value ed Endpoint\_Actual\_Value.

Dato che gli indici possono essere partizionati, è disponibile una vista del dizionario dati USER\_IND\_PARTITIONS. Le colonne contenute in USER\_IND\_PARTITIONS possono essere suddivise in tre categorie, come mostrato nella Tabella 37.15.

Le colonne presenti in USER\_IND\_PARTITIONS riproducono quelle contenute in USER\_INDEXES, con alcune modifiche alle colonne Identificazione. Le colonne Identificazione per gli indici partizionati mostrano il nome della partizione, il suo valore superiore e la posizione della partizione all'interno della tabella. Le colonne "correlate allo spazio" sono popolate quando la partizione viene analizzata (si veda il comando `analyze` nel Capitolo 42). Le colonne "correlate allo spazio" descrivono l'allocazione dello spazio per l'indice. Per informazioni sui parametri di memorizzazione, si consulti il paragrafo Memorizzazione del Capitolo 42.

Se una partizione possiede delle partizioni secondarie, tramite le viste del dizionario dati si potranno visualizzare i dettagli relativi a esse. USER\_IND\_SUBPARTITIONS contiene le colonne "correlate allo spazio" e "correlate alle statistiche" di USER\_IND\_PARTITIONS, insieme ad alcune colonne necessarie per identificare la partizione secondaria (Subpartition\_Name e

**Tabella 37.15** Colonne di USER\_IND\_PARTITIONS.

IDENTIFICAZIONE	CORRELATE ALLO SPAZIO	CORRELATE ALLE STATISTICHE
Index_Name	Tablespace_Name	Blevel
Composite	Ini_Trans	Leaf_Blocks
Partition_Name	Max_Trans	Distinct_Keys
Subpartition_Count	Initial_Extent	Avg_Leaf_Blocks_Per_Key
High_Value	Next_Extent	Avg_Data_Blocks_Per_Key
High_Value_Length	Min_Extent	Clustering_Factor
Partition_Position	Max_Extent	Num_Rows
Status	Pct_Increase	Sample_Size
Logging	Pct_Free	Last_Analyzed
Buffer_Pool	Freelists	User_Stats
Compression	Freelist_Groups	Pct_Direct_Access
Domidx_Opstatus		Global_Stats
Parameters		

Subpartition\_Position). Analogamente, USER\_TAB\_SUBPARTITIONS contiene le colonne correlate allo spazio e correlate alle statistiche di USER\_TAB\_PARTITIONS, insieme alle colonne Subpartition\_Name e Subpartition\_Position. Pertanto è possibile stabilire le definizioni di spazio per tutte le partizioni e le partizioni secondarie.

Come mostrato in precedenza in questo paragrafo, è possibile eseguire una query sulle viste del dizionario dati USER\_PART\_COL\_STATISTICS e USER\_PART\_HISTOGRAMS per ottenere le informazioni statistiche riguardanti le partizioni. Per le partizioni secondarie, si può eseguire una query su USER\_SUBPART\_COL\_STATISTICS e USER\_SUBPART\_HISTOGRAMS, le cui strutture ricordano quelle delle viste per le statistiche di partizione. Per visualizzare le colonne delle chiavi di partizioni secondarie, si può eseguire una query su USER\_SUBPART\_KEY\_COLUMNS, la cui struttura di colonna è identica a quella di USER\_PART\_KEY\_COLUMNS.

## **Spazio libero: USER\_FREE\_SPACE**

Oltre a visualizzare lo spazio utilizzato, è possibile effettuare query sul dizionario dati per sapere quanto spazio è attualmente contrassegnato come "libero". La vista USER\_FREE\_SPACE elenca gli extent liberi in tutte le tablespace accessibili all'utente. La vista elenca per Tablespace\_Name il File\_ID, il Block\_ID e il numero di file relativo del punto iniziale dell'extent libero. La dimensione dell'extent libero è visualizzata sia in byte sia in blocchi. DBA\_FREE\_SPACE viene utilizzata spesso dai DBA per controllare la quantità di spazio libero disponibile e il suo grado di frammentazione.

### **37.11 Utenti e privilegi**

Gli utenti e i relativi privilegi sono registrati all'interno del dizionario dati. Nei paragrafi seguenti, viene descritta l'esecuzione di query sul dizionario dati per ottenere informazioni sugli account degli utenti, sui limiti di risorse e sui privilegi degli utenti.

#### **Utenti: USER\_USERS**

Per visualizzare informazioni sul proprio account, è possibile eseguire una query su USER\_USERS. Questa vista include Username, User\_ID (un numero assegnato dal database), Default\_Tablespace, Temporary\_Tablespace e la data Created (data in cui è stato creato l'account). La colonna Account\_Status in USER\_USERS visualizza lo stato dell'account, precisando se questo è bloccato, sbloccato (OPEN) oppure scaduto. Se l'account è bloccato, la colonna Lock\_Date visualizza la data in cui l'account è stato bloccato, mentre la colonna Expiry\_Date visualizza la data di scadenza. Inoltre è possibile eseguire una query sulle informazioni relative al gruppo di consumer di risorse (Initial\_Rsrc\_Consumer\_Group) assegnato dal DBA e al valore di External\_Name.

ALL\_USERS contiene solo le colonne Username, User\_ID e Created tratte da USER\_USERS, ma elenca queste informazioni per tutti gli account del database. La vista ALL\_USERS si rivela utile quando è necessario conoscere quali nomi utente sono disponibili (per esempio, durante l'esecuzione di comandi grant). La vista DBA\_USERS contiene tutte le colonne presenti in USER\_USERS, più altre due colonne: Password (la password cifrata per l'account) e Profile (il profilo risorse dell'utente). DBA\_USERS elenca queste informazioni per tutti gli utenti del database.

## Limiti delle risorse: USER\_RESOURCE\_LIMITS

In Oracle, i *profili* consentono di imporre limiti alla quantità di risorse di sistema e del database disponibili per un utente. Se in un database non sono stati creati profili, viene utilizzato il profilo predefinito, che specifica risorse illimitate per tutti gli utenti. Le risorse che possono essere limitate sono descritte nella voce *create profile* del Capitolo 42. I profili impongono misure di sicurezza aggiuntive, come date di scadenza per gli account e una lunghezza minima per le password.

Per visualizzare i limiti attivi per la sessione corrente è possibile effettuare una query su *USER\_RESOURCE\_LIMITS*. Le colonne di questa vista sono:

Resource_Name	Nome della risorsa (per esempio SESSIONS_PER_USER).
Limit	Limite imposto per tale risorsa.

La vista *USER\_PASSWORD\_LIMITS* descrive i parametri del profilo password relativo all'utente. In essa sono presenti le stesse colonne contenute in *USER\_RESOURCE\_LIMITS*.

Per questa vista non esistono versioni “ALL” o “DBA”; ed è limitata alla sessione in corso dell'utente. Per visualizzare i costi associati a ciascuna risorsa disponibile, è possibile eseguire una query sulla vista *RESOURCE\_COST*. IDBA sono in grado di accedere alla vista *DBA\_PROFILES* per visualizzare i limiti imposti per le risorse di tutti i profili. La colonna *Resource\_Type* di *DBA\_PROFILES* indica se il profilo risorse è un profilo ‘PASSWORD’ o ‘KERNEL’.

## Privilegi di tabella: USER\_TAB\_PRIVS

Per visualizzare le concessioni per cui si è il concessionario, il concedente o il proprietario dell'oggetto, occorre effettuare una query su *USER\_TAB\_PRIVS* (privilegi di tabella dell'utente). Oltre alle colonne *Grantee*, *Grantor* e *Owner*, questa vista contiene le colonne *Table\_Name*, *Privilege* e un flag (impostato su ‘YES’ o ‘NO’) che indica se il privilegio è stato concesso con opzione amministrativa (*Grantable*).

*USER\_TAB\_PRIVS\_MADE* visualizza i record di *USER\_TAB\_PRIVS* di cui l'utente è proprietario (e pertanto non contiene una colonna *Owner*). *USER\_TAB\_PRIVS\_REC'D* (privilegi di tabella dell'utente ricevuti) visualizza i record di *USER\_TAB\_PRIVS* per i quali l'utente è concessionario (e pertanto non contiene una colonna *Grantee*). Dato che *USER\_TAB\_PRIVS\_MADE* e *USER\_TAB\_PRIVS\_REC'D* sono semplici sottoinsiemi di *USER\_TAB\_PRIVS*, è possibile emulare il loro funzionamento effettuando una query su *USER\_TAB\_PRIVS* con una clausola *where* adeguata per visualizzare il sottoinsieme desiderato.

Per *USER\_TAB\_PRIVS*, *USER\_TAB\_PRIVS\_MADE* e *USER\_TAB\_PRIVS\_REC'D* sono disponibili le versioni “ALL”. Esse elencano gli oggetti per i quali l'utente o PUBLIC sono il concessionario o il concedente. Esiste una versione “DBA” di *USER\_TAB\_PRIVS*, denominata *DBA\_TAB\_PRIVS*, che elenca tutti i privilegi di oggetto concessi a tutti gli utenti del database. *DBA\_TAB\_PRIVS* e *ALL\_TAB\_PRIVS* hanno le stesse definizioni delle colonne di *USER\_TAB\_PRIVS*.

## Privilegi di colonna: USER\_COL\_PRIVS

Oltre a concedere privilegi sulle tabelle, è anche possibile concedere privilegi a livello delle colonne. Per esempio, è possibile concedere agli utenti la capacità di aggiornare solo determinate colonne in una tabella. Per ulteriori informazioni, si rimanda al Capitolo 19 e alla voce *grant* nel Capitolo 42.

Le viste del dizionario dati adibite alla visualizzazione dei privilegi di colonna sono pressoché identiche nel disegno alle viste per i privilegi di tabella, descritte nel paragrafo precedente. In ogni vista COL esiste una colonna in più, Column\_Name, mentre è assente la colonna Hierarchy. USER\_COL\_PRIVS è analoga a USER\_TAB\_PRIVS, USER\_COL\_PRIVS\_MADE è analoga a USER\_TAB\_PRIVS\_MADE e USER\_COL\_PRIVS\_RECD è analoga a USER\_TAB\_PRIVS\_RECD. Inoltre, le tabelle seguenti hanno una colonna Gerarchia che le tabelle COL\_PRIVS correlate non possiedono: USER\_TAB\_PRIVS, USER\_TAB\_PRIVS\_MADE e USER\_TAB\_PRIVS\_RECD.

Per tutte le viste dei privilegi di colonna sono disponibili le versioni “ALL”. DBA\_COL\_PRIVS elenca tutti i privilegi di colonna concessi agli utenti del database (proprio come DBA\_TAB\_PRIVS elenca tutti i privilegi di tabella concessi agli utenti).

### **Privilegi di sistema: USER\_SYS\_PRIVS**

La vista USER\_SYS\_PRIVS elenca i privilegi di sistema concessi all’utente. Le colonne sono Username, Privilege e Admin\_Option (un flag impostato a ‘YES’ o ‘NO’ per indicare se il privilegio è stato concesso with admin option). Tutti i privilegi di sistema concessi direttamente a un utente sono visualizzati tramite questa vista. I privilegi di sistema concessi a un utente in grazia di un ruolo non sono visualizzati nello stesso elenco. La query seguente mostra dell’output di esempio per l’account PRATICA:

```
select * from USER_SYS_PRIVS;
```

USERNAME	PRIVILEGE
PRATICA	CREATE DIMENSION
PRATICA	UNLIMITED TABLESPACE

Non esiste una versione “ALL” per questa vista. Per visualizzare i privilegi di sistema concessi a tutti gli utenti del database, è necessaria una query su DBA\_SYS\_PRIVS, che ha le stesse definizioni delle colonne di USER\_SYS\_PRIVS.

### **37.12 Ruoli**

Oltre ai privilegi concessi direttamente agli utenti, è possibile raggruppare in ruoli gruppi ben determinati di privilegi. I ruoli possono essere concessi a utenti o ad altri ruoli e possono essere costituiti sia da privilegi di oggetto, sia da privilegi di sistema. Per informazioni sull’impiego e la gestione dei ruoli, si rimanda al Capitolo 19.

Per visualizzare i ruoli concessi a un utente, si dovrà effettuare una query sulla vista del dizionario dati USER\_ROLE\_PRIVS. In essa verrà visualizzato anche qualsiasi ruolo concesso al pubblico (PUBLIC). Le colonne disponibili per USER\_ROLE\_PRIVS sono le seguenti:

---

Username	Nome utente (può essere anche ‘PUBLIC’).
Granted_Role	Nome del ruolo concesso all’utente.
Admin_Option	Un flag per indicare se il ruolo è stato concesso with admin option oppure no (‘YES’ o ‘NO’).
Default_Role	Un flag per indicare se si tratta del ruolo predefinito dell’utente (‘YES’ o ‘NO’).
OS_Granted	Un flag per indicare se il sistema operativo viene usato per gestire i ruoli (‘YES’ o ‘NO’).

---

Per elencare tutti i ruoli disponibili nel database, è necessario disporre dell'autorità DBA; quindi, si può eseguire una query su DBA\_ROLES. DBA\_ROLE\_PRIVS elenca gli assegnamenti di questi ruoli a tutti gli utenti del database.

I ruoli possono ricevere tre tipi diversi di concessioni e a ciascuna di esse corrisponde una vista del dizionario dati:

Concessioni di tabella/colonna	ROLE_TAB_PRIVS. Simile a USER_TAB_PRIVS e USER_COL_PRIVS, eccettuata la presenza di una colonna Role al posto della colonna Grantee.
Privilegi di sistema	ROLE_SYS_PRIVS. Simile a USER_SYS_PRIVS, eccettuata la presenza di una colonna Role al posto della colonna Username.
Concessioni di ruolo	ROLE_ROLE_PRIVS. Elenca tutti i ruoli concessi ad altri ruoli.

**NOTA** *Se l'utente non è un DBA, queste viste del dizionario dati mostrano solo i ruoli garantiti all'utente stesso.*

Oltre a queste viste ne esistono altre due, ciascuna con un'unica colonna, che elencano i privilegi e i ruoli abilitati per la sessione in corso:

SESSION_PRIVS	La colonna Privilege elenca tutti i privilegi di sistema disponibili per la sessione, sia concessi direttamente, sia tramite ruoli.
SESSION_ROLES	La colonna Role elenca tutti i ruoli attualmente abilitati per la sessione.

SESSION\_PRIVS e SESSION\_ROLES sono disponibili per tutti gli utenti.

**NOTA** *Per informazioni sull'attivazione e la disattivazione dei ruoli e sull'impostazione di ruoli predefiniti si rimanda al Capitolo 19.*

### 37.13 Audit

In qualità di utente senza autorità di DBA all'interno di un database Oracle, non è possibile abilitare le caratteristiche di audit del database stesso. Se però l'audit è stato abilitato, esisteranno delle viste del dizionario dati utilizzabili da chiunque per visualizzare la procedura di audit.

Per le procedure di audit sono disponibili molte viste del dizionario dati. La maggior parte di queste viste si basa su un'unica tabella di procedura di audit nel database (SYS.AUD\$). La vista più generica che ci sia a disposizione per la procedura di audit è USER\_AUDIT\_TRAIL. Le colonne di questa vista sono descritte nella Tabella 37.16. Dato che USER\_AUDIT\_TRAIL mostra i record di audit per molti tipi diversi di azioni, molte delle colonne potranno non essere applicabili a tutte le righe. La versione "DBA" di questa vista, DBA\_AUDIT\_TRAIL, elenca tutte le voci dalla tabella della procedura di audit. La vista USER\_AUDIT\_TRAIL elenca solo le voci importanti per l'utente.

Come si vede nella Tabella 37.16, la gamma di possibilità di audit disponibili è piuttosto vasta (per un elenco completo si consulti la voce relativa al comando audit nel Capitolo 42). A ciascun tipo di audit è possibile accedere tramite la relativa vista del dizionario dati. Ecco le viste disponibili:

---

USER_AUDIT_OBJECT	Per istruzioni relative agli oggetti.
USER_AUDIT_SESSION	Per connessioni e disconnessioni.
USER_AUDIT_STATEMENT	Per comandi grant, revoke, audit, noaudit e alter system eseguiti dall'utente.

---

Per ciascuna delle viste “USER” elencate precedentemente esiste una versione “DBA” che visualizza tutti i record di procedura di audit che rientrano nella categoria della vista.

Per visualizzare le opzioni di audit attualmente abilitate per i propri oggetti, è necessario eseguire una query su USER\_OBJ\_AUDIT\_OPTS. In USER\_OBJ\_AUDIT\_OPTS, per ciascun oggetto menzionato sono elencate le opzioni di audit relative a ciascun comando che può essere eseguito sull’oggetto stesso (identificato dalle colonne Object\_Name e Object\_Type). I nomi delle colonne di USER\_OBJ\_AUDIT\_OPTS corrispondono alle prime tre lettere del comando (per esempio, Alt per alter, Upd per update). Ciascuna colonna registra se il comando è stato sottoposto ad audit per un determinato oggetto, quando il comando stesso è completato con successo (‘S’), non è completato (‘U’) o in entrambi i casi. Le opzioni di audit predefinite attivate per eventuali oggetti nuovi nel database possono essere visualizzate tramite la vista ALL\_DEF\_AUDIT\_OPTS, che utilizza le stesse convenzioni per l’assegnazione dei nomi alle colonne utilizzate da USER\_OBJ\_AUDIT\_OPTS.

I comandi che possono essere sottoposti ad audit vengono memorizzati in una tabella di riferimento, denominata AUDIT\_ACTIONS, che presenta due colonne: Action (codice numerico per l’azione) e Name (nome dell’azione/comando). Action e Name corrispondono alle colonne Action e Action\_Name della vista USER\_AUDIT\_TRAIL.

I DBA possono utilizzare molte altre viste di audit prive di versioni “USER” corrispondenti, tra cui DBA\_AUDIT\_EXISTS, DBA\_PRIV\_AUDIT\_OPTS, DBA\_STMT\_AUDIT\_OPTS e STMT\_AUDIT\_OPTION\_MAP. Per ulteriori informazioni su queste viste riservate ai DBA, si rimanda alla *Oracle Server Administrator’s Guide*.

## 37.14 Varie

Oltre alle viste del dizionario dati descritte nei paragrafi precedenti di questo capitolo, all’interno del dizionario dati possono essere disponibili numerose viste e tabelle di vario tipo. Queste viste e tabelle comprendono le viste riservate ai DBA e la tabella utilizzata quando si esegue il comando explain plan. Nei paragrafi seguenti, sono descritti brevemente tutti i vari tipi di viste.

## 37.15 Monitoraggio: le tabelle di prestazioni dinamiche V\$

Le viste che consentono il monitoraggio delle prestazioni ambientali del database prendono il nome di *viste statistiche di sistema*. Le tabelle statistiche di sistema, dette anche tabelle di prestazioni dinamiche, sono comunemente note come tabelle V\$ (pronunciate “V-Dollaro”) in quanto iniziano tutte con la lettera “V” seguita dal simbolo del dollaro (\$).

Le definizioni e l’uso delle colonne all’interno delle viste di monitoraggio sono soggetti a modifiche con ogni nuova versione del database. La corretta interpretazione dei risultati di query ad hoc eseguite su viste solitamente richiede un riferimento alla *Oracle9i Database Administrator’s Guide*. Normalmente le tabelle V\$ sono utilizzate solo dai DBA. Per dettagli sull’impiego delle viste V\$, si consulti l’*Oracle9i Database Reference* e l’*Oracle9i DBA Handbook*.

**Tabella 37.16** Colonne di USER\_AUDIT\_TRAIL.

NOME COLONNA	DESCRIZIONE
OS_Username	Account di sistema operativo dell'utente controllato.
Username	Nome utente ORACLE dell'utente revisionato.
UserHost	ID numerico per l'istanza utilizzata dall'utente controllato.
Terminal	Identificatore di terminale del sistema operativo dell'utente.
TimeStamp	Data e ora di creazione del record di audit.
Owner	Proprietario dell'oggetto influenzato da un'azione (per l'audit di azioni).
Obj_Name	Nome dell'oggetto influenzato da un'azione (per l'audit di azioni).
Action	Codice numerico per l'azione controllata.
Action_Name	Nome dell'azione controllata.
New_Owner	Proprietario dell'oggetto menzionato nella colonna New_Name.
New_Name	Nuovo nome di un oggetto sottoposto a rename.
Obj_Privilege	Privilegio di oggetto sottoposto a grant o revoke.
Sys_Privilege	Privilegio di sistema sottoposto a grant o revoke.
Admin_Option	Flag che indica se il ruolo o privilegio di sistema è stato sottoposto a grant with admin option ('Y' o 'N').
Grantee	Nome utente specificato in un comando grant o revoke.
Audit_Option	Opzioni di revisione impostate mediante un comando audit.
Ses_Actions	Stringa di caratteri che serve come riassunto della sessione e registra i successi e gli insuccessi delle varie azioni.
Logoff_Time	Data e ora di disconnessione dell'utente.
Logoff_LRead	Numero di letture logiche eseguite durante la sessione.
Logoff_PRead	numero di letture fisiche eseguite durante la sessione.
Logoff_LWrite	Numero di scritture logiche eseguite durante la sessione.
Logoff_DLock	Numero di deadlock individuali durante la sessione.
Comment_Text	Commento testuale sulla voce relativa alla procedura di audit.
SessionID	ID numerico della sessione.
EntryID	ID numerico per la voce relativa alla procedura di auditing.
StatementID	ID numerico per ciascun comando eseguito.
ReturnCode	Codice di restituzione per ciascun comando eseguito. Se il comando è stato completato con successo, il codice ReturnCode è 0.
Priv_Used	Privilegio di sistema utilizzato per eseguire l'azione.
Client_ID	ID del client.
Session_CPU	CPU della sessione utilizzata.

## CHAINED\_ROWS

Quando una riga non rientra più all'interno del blocco di dati in cui è memorizzata la propria intestazione, può memorizzare i dati restanti in un blocco o in un gruppo di blocchi differente. Una simile riga viene detta *concatenata*. Le righe concatenate possono provocare una riduzione delle prestazioni a causa dell'aumentato numero di blocchi che dovranno essere letti allo scopo di leggere una singola riga.

Il comando `analyze` consente di generare un elenco delle righe concatenate all'interno di una tabella. L'elenco delle righe concatenate può essere memorizzato in una tabella di nome CHAINED\_ROWS. Per creare la tabella CHAINED\_ROWS nel proprio schema, occorre eseguire lo script `utlchain.sql` (che normalmente si trova nella sottodirectory `/rdbms/admin` della directory home di Oracle).

Per popolare la tabella CHAINED\_ROWS, si utilizzi la clausola `list chained rows into` del comando `analyze`, come mostrato nel listato seguente:

```
analyze TABLE COMPLEANNO list chained rows into CHAINED_ROWS;
```

La tabella CHAINED\_ROWS elenca le colonne `Owner_Name`, `Table_Name`, `Cluster_Name` (se la tabella si trova in un cluster), `Partition_Name` (se la tabella è partizionata), `Subpartition_Name` (se la tabella contiene delle partizioni secondarie) `Head_RowID` (RowID della riga) e una colonna `Timestamp` che visualizza l'ultima volta in cui la tabella o il cluster sono stati analizzati. È possibile eseguire una query sulla tabella che si basa sui valori di `Head_RowID` in CHAINED\_ROWS, come mostrato nell'esempio seguente:

```
select * from COMPLEANNO
where RowID in
  (select Head_RowID
   from CHAINED_ROWS
   where Table_Name = 'COMPLEANNO');
```

Se la riga concatenata è di lunghezza ridotta, allora si potrà rimuovere la concatenazione eliminando e reinserendo la riga.

## PLAN\_TABLE

Quando si mettono a punto le istruzioni SQL, è possibile stabilire i passaggi che l'ottimizzatore dovrà seguire per eseguire la query. Per visualizzare il percorso di query, per prima cosa è necessario creare nel proprio schema una tabella di nome PLAN\_TABLE. Lo script utilizzato per creare questa tabella si chiama `utlxplan.sql` e normalmente si trova nella sottodirectory `/rdbms/admin` della directory home di Oracle.

Dopo aver creato la tabella PLAN\_TABLE nel proprio schema, si può utilizzare il comando `explain plan`, che genera dei record in PLAN\_TABLE, contrassegnato con il valore `Istruzione_ID` specificato per la query che si intende spiegare:

```
explain plan
set Statement_ID = 'MIOTEST'
for
select * from COMPLEANNO
where Cognome like 'S%';
```

Le colonne `ID` e `Parent_ID` di PLAN\_TABLE stabiliscono la gerarchia dei passaggi (Operations) che l'ottimizzatore seguirà mentre esegue la query. Per ulteriori informazioni

sull'ottimizzatore di Oracle e l'interpretazione dei record di PLAN\_TABLE, si rimanda al Capitolo 38.

## Interdipendenze: USER\_DEPENDENCIES e IDEPTREE

Gli oggetti all'interno dei database Oracle possono essere interdipendenti. Per esempio, una stored procedure può dipendere da una tabella, o un package può dipendere dal corpo di un package. Quando un oggetto all'interno del database viene modificato, qualsiasi oggetto procedurale che dipenda dall'oggetto dovrà essere ricompilato. La ricompilazione può avvenire automaticamente al momento dell'esecuzione (con una conseguente penalizzazione delle prestazioni) o manualmente (per ulteriori informazioni sulla compilazione degli oggetti procedurali si rimanda al Capitolo 29).

Per aiutare a tenere traccia di queste interdipendenze sono disponibili due gruppi di viste del dizionario dati. La prima è USER\_DEPENDENCIES, che elenca tutte le dipendenze *dirette* tra gli oggetti. Tale elenco scende però di un solo livello lungo l'albero delle interdipendenze. Per valutare completamente le interdipendenze, è necessario creare nel proprio schema oggetti ricorsivi di segnalazione delle dipendenze. Per creare questi oggetti, occorre eseguire lo script utldtree.sql (normalmente situato nella sottodirectory /rdbms/admin della directory home di Oracle). Questo script crea due oggetti su cui è possibile eseguire una query: DEPTREE e IDEPTREE. I due oggetti contengono informazioni identiche, ma IDEPTREE è indentato in base ai valori della pseudocolonna Level, quindi è più facile da leggere e interpretare.

## Viste riservate ai DBA

Dato che questo capitolo è dedicato agli sviluppatori e agli utenti finali, le viste del dizionario dati riservate ai DBA non verranno analizzate. Queste viste sono utilizzate per fornire informazioni su transazioni distribuite, dispute di lock, segmenti di rollback e altre funzioni interne del database. Per informazioni sull'uso delle viste riservate ai DBA, si consulti la *Oracle9i Database Administrator's Guide*.

## Oracle Label Security

Gli utenti di Oracle Label Security sono in grado di visualizzare altre viste del dizionario dati, tra cui ALL\_SA\_GROUPS, ALL\_SA\_POLICIES, ALL\_SA\_USERS e ALL\_SA\_USER\_PRIVS. Per ulteriori informazioni sull'uso di queste viste, si consulti la *Oracle Label Security Administrator's Guide*.

## Viste per il caricamento diretto in SQL\*Loader

Per gestire l'opzione di caricamento diretto all'interno di SQL\*Loader, Oracle mette a disposizione numerose viste del dizionario dati. Generalmente, su tali viste si effettuano query solo a scopi di debug, dietro richiesta del Supporto clienti Oracle. L'opzione di caricamento diretto di SQL\*Loader è descritta alla voce "SQLLDR" nel Capitolo 42; le viste di supporto del dizionario dati sono le seguenti:

- LOADER\_COL\_INFO
- LOADER\_CONSTRAINT\_INFO
- LOADER\_FILE\_TS

- LOADER\_PARAM\_INFO
- LOADER\_PART\_INFO
- LOADER\_REF\_INFO
- LOADER\_TAB\_INFO
- LOADER\_TRIGGER\_INFO

Per dettagli sull'uso di queste viste, si rimanda allo script catldr.sql script, in genere situati nella sottodirectory /rdbms/admin della directory home di Oracle.

## Viste NLS (National Language Support)

Tre viste del dizionario dati vengono utilizzate per visualizzare informazioni sui parametri NLS (National Language Support) attualmente abilitati per il database. I valori non standard per i parametri NLS (come NLS\_DATE\_FORMAT e NLS\_SORT) possono essere impostati tramite il file dei parametri del database oppure con il comando alter session (per ulteriori informazioni sulle impostazioni NLS, si consulti il comando alter session nel Capitolo 42). Per visualizzare le impostazioni NLS correnti per la propria sessione, istanza e database, è necessario eseguire una query rispettivamente su NLS\_SESSION\_PARAMETERS, NLS\_INSTANCE\_PARAMETERS e NLS\_DATABASE\_PARAMETERS.

## Librerie

Le routine PL/SQL (Capitolo 27) sono in grado di eseguire programmi C esterni. Per visualizzare le librerie di programmi C esterni possedute, è possibile effettuare una query su USER\_LIBRARIES, che visualizza il nome della libreria (Library\_Name), il file associato (File\_Spec), specifica se la libreria può essere caricata dinamicamente (Dynamic) e visualizza lo stato della libreria stessa (Status). Sono inoltre disponibili le viste ALL\_LIBRARIES e DBA\_LIBRARIES. Tali viste contengono una colonna in più, Owner, che indica il proprietario della libreria. Per ulteriori informazioni sulle librerie si consulti il paragrafo dedicato a create library nel Capitolo 42.

## Servizi eterogenei

Per supportare la gestione di servizi eterogenei, Oracle mette a disposizione sedici viste del dizionario dati. Tutte le viste di questa categoria iniziano con le lettere HS, invece di DBA. In generale, queste viste servono principalmente ai DBA. Per dettagli sulle viste HS, si consulti *Oracle9i Database Reference*.

## Tipi di indici e operatori

Gli operatori e i tipi di indici sono strettamente correlati. Per creare un nuovo operatore e definire le associazioni, è possibile ricorrere al comando create operator. Nei tipi di indici e nelle istruzioni SQL è possibile fare riferimento agli operatori. Gli operatori, a loro volta, fanno riferimento a funzioni, package, tipi e altri oggetti definiti dall'utente.

Per visualizzare i valori Owner, Operator\_Name e Number\_of\_Binds di ciascun operatore è possibile eseguire una query sulla vista USER\_OPERATORS. Alle informazioni complementari degli operatori si può accedere tramite USER\_OPANCILLARY, mentre per visualizzare gli ar-

gomenti degli operatori si può eseguire una query su USER\_OPARGUMENTS. Per visualizzare le associazioni di un operatore, è sufficiente eseguire una query su USER\_OPBINDINGS.

USER\_INDEXTYPE\_OPERATORS elenca gli operatori supportati dai vari tipi di indici. I tipi di indici, a loro volta, vengono visualizzati tramite USER\_INDEXTYPES. Per tutte le viste degli operatori e dei tipi di indici esistono le versioni “ALL” e “DBA”.

## Strutture

Quando si usano strutture memorizzate, è possibile recuperare il nome e i dettagli delle strutture tramite le viste del dizionario dati USER\_OUTLINES. Per visualizzare i suggerimenti che costituiscono la struttura, si esegua una query su USER\_OUTLINE\_HINTS. Per USER\_OUTLINES e USER\_OUTLINE\_HINTS sono disponibili le versioni “ALL” e “DBA”.

## Capitolo 38

# Guida all'ottimizzatore di Oracle

- 38.1 **Quale ottimizzatore?**
- 38.2 **Operazioni di accesso alle tabelle**
- 38.3 **Operazioni che utilizzano gli indici**
- 38.4 **Operazioni che gestiscono insiemi di dati**
- 38.5 **Operazioni che eseguono join**
- 38.6 **Visualizzazione del percorso di esecuzione**
- 38.7 **Operazioni varie**
- 38.8 **Conclusioni**

**N**el modello relazionale la dislocazione fisica dei dati non è importante. Anche in Oracle la dislocazione fisica dei dati e l'operazione utilizzata per recuperarli non sono importanti, finché il database non necessita di recuperare questi dati. Se si esegue una query sul database, è necessario prestare attenzione alle operazioni eseguite da Oracle per recuperare e manipolare i dati. Quanto meglio si comprende il percorso esecutivo utilizzato da Oracle per effettuare la query, tanto meglio si riuscirà a gestire e mettere a punto la query stessa.

In questo capitolo vengono analizzate le operazioni utilizzate da Oracle per effettuare la query dei dati e per elaborarli, dal punto di vista dell'utente. Si inizia con le operazioni per accedere alle tabelle, seguite dalle operazioni per accedere agli indici, dalle operazioni per impostare i dati, dai join e dalle operazioni di altro tipo. Per ciascun tipo di operazione vengono fornite informazioni importanti per quanto riguarda la messa a punto, in modo da aiutare l'utente a utilizzarla nel modo più efficiente ed efficace possibile.

Questo capitolo concentra la sua attenzione sulle operazioni che Oracle segue quando esegue un'istruzione SQL. Se si cerca di mettere a punto un'applicazione, prima di esaminare il codice SQL si dovrebbe valutare l'architettura e l'ambiente operativo dell'applicazione per stabilire se sono consoni ai requisiti degli utenti. Un'applicazione che esegue un gran numero di query in una rete lenta solo per visualizzare una schermata di dati viene percepita come lenta anche se la porzione attiva del database è veloce; la messa a punto di SQL in una situazione simile può aiutare a migliorare le prestazioni.

Prima di iniziare la messa a punto delle query, è necessario decidere quale ottimizzatore utilizzare.

## 38.1 Quale ottimizzatore?

L'ottimizzatore di Oracle presenta due modalità principali di funzionamento: una basata sui costi e l'altra basata sulle regole. Per impostare lo scopo dell'ottimizzatore, si può specificare CHOOSE (per la modalità basata sui costi) oppure RULE (per quella basata sulle regole) per il parametro OPTIMIZER\_MODE nel file dei parametri di inizializzazione del database. È possi-

bile riassegnare le operazioni predefinite dell'ottimizzatore a livello di query e di sessione, come mostrato più avanti in questo capitolo.

**NOTA** *A partire da Oracle9i è possibile memorizzare i parametri in un file dei parametri di sistema, che sostituisce il file dei parametri init.ora impiegato nelle versioni precedenti.*

Impostando OPTIMIZER\_MODE al valore RULE, si invoca l'*ottimizzatore basato su regole (RBO)* che valuta i possibili percorsi esecutivi e fa una stima dei percorsi esecutivi alternativi in base a una serie di regole di sintassi. In genere le applicazioni nuove usano raramente l'RBO, che si trova principalmente in applicazioni sviluppate e messe a punto per le versioni precedenti di Oracle.

Impostando OPTIMIZER\_MODE al valore CHOOSE, si richiama l'*ottimizzatore basato sui costi (CBO)*. Si può utilizzare il comando analyze e il package DBMS\_STATS per generare statistiche relative agli oggetti del proprio database. Le statistiche includono il numero di righe in una tabella e il numero di chiavi distinte presenti all'interno di un indice. Basandosi su queste statistiche, il CBO valuta il costo dei percorsi esecutivi disponibili e seleziona il percorso che ha il costo relativo più basso. Se si utilizza il CBO, è opportuno accertarsi di avere analizzato i dati con una frequenza tale da consentire alle statistiche di rispecchiare accuratamente i dati presenti nel database. Se una query si riferisce sia a tabelle che sono state analizzate sia a tabelle che non sono state analizzate, il CBO seleziona dei valori per le statistiche mancanti, e può anche decidere di seguire un percorso esecutivo non adatto. Per migliorare le prestazioni, è consigliabile utilizzare l'RBO o il CBO in modo coerente nel database. Dato che il CBO supporta modifiche nei volumi dei dati e nella distribuzione dei medesimi, è opportuno favorirne l'uso.

Per usare il CBO, per prima cosa è necessario analizzare le tabelle e gli indici. Con il comando analyze si possono analizzare singole tabelle, indici, partizioni o cluster (per la sintassi completa di questo comando si rimanda al Capitolo 42). Durante la fase di analisi, è possibile eseguire la scansione dell'oggetto intero (con la clausola compute statistics) oppure di una parte dell'oggetto (con la clausola estimate statistics). In linea di massima, con un'analisi pari al 10-20% dell'oggetto si possono raccogliere statistiche adeguate, impiegando molto meno tempo di che quello che servirebbe per calcolare le statistiche. Ecco un semplice comando analyze:

```
analyze table BIBLIOTECA estimate statistics;
```

Terminata l'analisi di un oggetto, è possibile eseguire una query sulle colonne correlate alle statistiche delle viste del dizionario dati per visualizzare i dati così creati. Per una descrizione di queste viste e delle loro colonne correlate alle statistiche, si rimanda al Capitolo 37.

Il package DBMS\_STATS è un sostituto del comando analyze, e a partire da Oracle9i è anche il metodo consigliato. La procedura GATHER\_TABLE\_STATS in DBMS\_STATS richiede due parametri: il nome del proprietario dello schema e il nome della tabella; tutti gli altri parametri (come il nome delle partizioni e la percentuale di tabella per cui si deve eseguire la scansione con il metodo di valutazione delle statistiche) sono facoltativi. Il comando seguente raccoglie le statistiche per la tabella BIBLIOTECA nello schema PRATICA:

```
execute DBMS_STATS.GATHER_TABLE_STATS('PRATICA','BIBLIOTECA');
```

Altre procedure all'interno di DBMS\_STATS includono GATHER\_INDEX\_STATS (per gli indici), GATHER\_SCHEMA\_STATS (per tutti gli oggetti di uno schema), GATHER DATABASE\_STATS (per tutti gli oggetti di un database) e GATHER\_SYSTEM\_STATS (per le stati-

stiche di sistema). Le altre procedure contenute nel package DBMS\_STATS possono servire per effettuare la migrazione delle statistiche da un database in un altro, evitando così un nuovo calcolo delle statistiche per copie diverse delle stesse tabelle. Per ulteriori informazioni sul package DBMS\_STATS, si consulti il documento *Supplied PL/SQL Packages and Types Reference* di Oracle.

Gli esempi di questo paragrafo richiedono l'uso dell'ottimizzatore basato sui costi e che le tabelle e gli indici siano stati analizzati.

## 38.2 Operazioni di accesso alle tabelle

Esistono due operazioni per accedere direttamente alle righe di una tabella: la scansione completa della tabella e l'accesso alla tabella basato sul RowID. Per informazioni sulle operazioni che accedono alle righe di una tabella tramite i cluster, si rimanda al paragrafo “Query che utilizzano cluster” più avanti in questo capitolo.

### TABLE ACCESS FULL

Durante la scansione completa di una tabella vengono lette in sequenza tutte le righe della tabella. L'ottimizzatore chiama questo tipo di operazione TABLE ACCESS FULL. Per ottimizzare le prestazioni di una scansione completa di una tabella, Oracle legge più blocchi nel corso di ciascuna lettura effettuata sul database.

Una scansione completa di tabella viene utilizzata ogni volta che nella query non è presente alcuna clausola *where*. Per esempio, la query seguente seleziona tutte le righe dalla tabella BIBLIOTECA:

```
select * from BIBLIOTECA;
```

Per risolvere la query precedente, Oracle esegue una scansione completa sulla tabella BIBLIOTECA. Se questa tabella è di dimensioni ridotte l'operazione può risolversi in breve tempo, con un minimo impatto sulle prestazioni. Tuttavia, via via che le dimensioni della tabella BIBLIOTECA aumentano, crescono anche i costi dell'operazione in termini di prestazioni. Se più utenti eseguono scansioni complete della tabella BIBLIOTECA, allora i costi associati crescono ancora più velocemente.

Con una pianificazione adeguata, le scansioni complete di una tabella non rappresentano per forza un problema per le prestazioni. Si dovrebbe collaborare con i propri amministratori di database per essere certi che quest'ultimo sia stato configurato in modo tale da sfruttare caratteristiche come la Parallel Query Option e le letture di più blocchi. A meno che l'ambiente per le scansioni complete di tabelle non sia stato configurato correttamente, sarebbe opportuno controllarne l'uso.

**NOTA** *In base ai dati selezionati, l'ottimizzatore potrebbe decidere di ricorrere a una scansione completa di un indice al posto di una scansione completa di tabella.*

È possibile visualizzare il percorso esecutivo di Oracle scelto tramite una caratteristica denominata “explain plan.” Più avanti in questo capitolo, si spiegherà in che modo creare degli explain plan. Per questo esempio, l'explain plan presenta l'aspetto seguente:

### Execution Plan

```

0   SELECT STATEMENT Optimizer=CHOOSE (Cost=1 Card=31 Bytes=1209
)
1   0   TABLE ACCESS (FULL) OF 'BIBLIOTECA' (Cost=1 Card=31 Bytes=1
209)

```

L’operazione TABLE ACCESS (FULL) riportata nell’explain plan mostra che l’ottimizzatore ha deciso di eseguire una scansione completa della tabella BIBLIOTECA. Ogni passaggio ha un proprio ID (in questo caso, 0 e 1); il passaggio 0 corrisponde alla fase di restituzione dei dati all’utente. I vari passaggi possono avere dei passaggi principali (in questo esempio, il passaggio 1 fornisce i suoi dati al passaggio 0, e nell’elenco compare che possiede un ID principale 0). Il grado di complessità degli explain plan cresce in maniera direttamente proporzionale a quello delle query. Le operazioni impiegate da Oracle sono l’argomento principale di questo capitolo; creando un explain plan è possibile controllarne le varie fasi. Per semplificare l’analisi di questo argomento, la spiegazione dettagliata della creazione e dell’interpretazione di explain plan complessi verrà rimandata a un punto successivo del capitolo; per descrivere i passaggi principali del percorso esecutivo e il flusso dei dati si ricorrerà a un metodo grafico.

## TABLE ACCESS BY ROWID

Per migliorare le prestazioni degli accessi alle tabelle, si possono utilizzare le operazioni Oracle che accedono alle righe in base ai loro valori di RowID. RowID registra la locazione fisica in cui è memorizzata la riga. Oracle utilizza gli indici per mettere in relazione i valori dei dati con i valori di RowID e così con le locazioni fisiche dei dati. Conoscendo il RowID di una riga, Oracle potrà usare l’operazione TABLE ACCESS BY ROWID per recuperare la riga in questione.

Quando è noto il RowID, si conosce anche l’esatta collocazione fisica della riga. Tuttavia, non è necessario memorizzare i valori di RowID delle righe; al contrario, si possono sfruttare gli indici per accedere alle informazioni riguardanti i RowID, come descritto nel paragrafo “Operazioni che utilizzano gli indici”. Dato che gli indici consentono un accesso rapido ai valori di RowID, aiutano a migliorare le prestazioni delle query che fanno uso di colonne indicizzate.

## Suggerimenti

All’interno di una query è possibile specificare suggerimenti che guidino il CBO nell’elaborazione della query stessa. Per specificare un suggerimento, si utilizza la sintassi mostrata nell’esempio che segue. Si inserisca la stringa seguente subito dopo la parola chiave select:

/\*+

Quindi si aggiunga un suggerimento, per esempio:

FULL(biblioteca)

Si termini il suggerimento con la stringa seguente:

\*/

I suggerimenti utilizzano la sintassi di Oracle per i commenti contenuti all’interno delle query, con l’aggiunta all’inizio del segno “+”. Nel corso di questo capitolo vengono descritti i suggerimenti rilevanti per ciascuna delle operazioni. Per quanto riguarda gli accessi alle tabelle

esistono due suggerimenti rilevanti: FULL e ROWID. L'opzione FULL comunica a Oracle di eseguire una scansione completa della tabella specificata (ovvero, l'operazione TABLE ACCESS FULL), come mostrato nel listato seguente:

```
select /*+ FULL(biblioteca) */ *
  from BIBLIOTECA
 where Titolo like "T%";
```

Se non si utilizzasse il suggerimento FULL per risolvere questa query, Oracle pianificherebbe normalmente l'impiego dell'indice di chiave primaria sulla colonna Titolo. Dato che la tabella è ancora piccola, una sua scansione completa non è dispendiosa. Quando le dimensioni della tabella incominceranno a crescere, probabilmente si opterà per un accesso basato sui valori di RowID.

L'opzione ROWID comunica all'ottimizzatore di utilizzare un'operazione TABLE ACCESS BY ROWID per accedere alle righe della tabella. In linea di massima, si dovrebbe ricorrere a questo tipo di operazione ogni volta che si ha la necessità di restituire velocemente le righe agli utenti e quando le tabelle hanno dimensioni grosse. Per utilizzare l'operazione TABLE ACCESS BY ROWID è necessario conoscere i valori di RowID o utilizzare un indice.

### 38.3 Operazioni che utilizzano gli indici

In Oracle esistono due tipi principali di indici: gli *indici unici*, in cui ogni riga della tabella indicizzata contiene un valore unico per la colonna o le colonne indicizzate, e gli *indici non unici*, in cui i valori indicizzati delle righe possono ripetersi. Le operazioni impiegate per leggere i dati dagli indici dipendono dal tipo di indice adottato e dal modo in cui viene scritta la query che accede all'indice.

Si consideri la tabella BIBLIOTECA:

```
create table BIBLIOTECA
(Titolo      VARCHAR2(100) primary key,
Editore     VARCHAR2(20),
NomeCategoria  VARCHAR2(20),
Classificazione  VARCHAR2(2),
constraint CATFK foreign key (NomeCategoria)
references CATEGORIA(NomeCategoria));
```

La colonna Titolo è la chiave primaria della tabella BIBLIOTECA, ovvero identifica in modo univoco ogni riga, e ogni attributo dipende dal valore di Titolo.

Ogni volta che si crea un vincolo di chiave primaria (PRIMARY KEY) o un vincolo unico (UNIQUE), Oracle genera un indice unico per impostare l'univocità dei valori nella colonna. Come definito dal comando `create table`, sulla tabella BIBLIOTECA verrà creato un vincolo di chiave primaria. All'indice che supporta la chiave primaria verrà assegnato un nome generato dal sistema, poiché il vincolo non è stato nominato in modo esplicito.

Si possono creare manualmente degli indici sulle altre colonne della tabella BIBLIOTECA. Per esempio, si potrebbe creare un indice non unico sulla colonna NomeCategoria con il comando `create index`:

```
create index BIBLIOTECA$CATEGORIA
  on BIBLIOTECA(NomeCategoria)
tablespace INDICI
compute statistics;
```

Ora, la tabella BIBLIOTECA possiede due indici: un indice unico sulla colonna Titolo, e un indice non unico sulla colonna NomeCategoria. Per la risoluzione di una query si potrebbero utilizzare uno o più indici, in base al modo in cui la query stessa viene scritta ed eseguita. In quanto parte del processo di creazione dell'indice, le statistiche a esso relative sono state raccolte tramite la clausola `compute statistics`. Dato che la tabella è già popolata con delle righe, non sarà necessario eseguire un comando separato per analizzare l'indice.

## INDEX UNIQUE SCAN

Per utilizzare un indice durante una query, questa dev'essere scritta in modo da consentire l'uso di un indice. Nella maggior parte dei casi, si consente all'ottimizzatore di utilizzare un indice attraverso la clausola `where` della query. Per esempio, la query seguente può utilizzare l'indice unico sulla colonna Titolo:

```
select *
  from BIBLIOTECA
 where Titolo = "INNUMERACY";
```

Internamente, l'esecuzione della query precedente viene suddivisa in due fasi. Per prima cosa, si accede all'indice della colonna Titolo tramite un'operazione INDEX UNIQUE SCAN. Dall'indice viene restituito il valore di RowID che corrisponde al titolo "INNUMERACY"; questo valore di RowID viene quindi utilizzato per eseguire una query sulla tabella BIBLIOTECA attraverso un'operazione TABLE ACCESS BY ROWID.

Se tutte le colonne selezionate dalla query fossero contenute nell'indice, Oracle non avrebbe bisogno di utilizzare l'operazione TABLE ACCESS BY ROWID; nel caso in cui i dati si trovassero nell'indice, questo rappresenterebbe tutto ciò che serve per soddisfare la query. Poiché in questo caso la query ha selezionato tutte le colonne dalla tabella BIBLIOTECA, e l'indice non conteneva tutte le colonne di questa tabella, è stato necessario ricorrere all'operazione TABLE ACCESS BY ROWID.

## INDEX RANGE SCAN

Quando si interroga il database in base a un intervallo di valori, o si effettua una query con un indice non unico, si utilizza un'operazione INDEX RANGE SCAN per eseguire la query sull'indice.

Si consideri nuovamente la tabella BIBLIOTECA, con un indice unico sulla colonna Titolo. Una query del tipo:

```
select Titolo
  from BIBLIOTECA
 where Titolo like 'M%';
```

restituirebbe tutti i valori di Titolo che iniziano con 'M'. Poiché la clausola `where` si riferisce alla colonna Titolo, l'indice di chiave primaria di questa colonna potrà servire a risolvere la query. Tuttavia, nella clausola `where` non è specificato un valore unico, bensì un intervallo di valori. Di conseguenza, si accede all'indice unico di chiave primaria tramite un'operazione INDEX RANGE SCAN. Poiché le operazioni INDEX RANGE SCAN richiedono la lettura di più valori dall'indice, risultano meno efficienti delle operazioni INDEX UNIQUE SCAN.

Nell'esempio precedente, la query ha selezionato solo la colonna Titolo. Dato che i valori della colonna Titolo sono memorizzati nell'indice di chiave primaria, che è in fase di analisi, non

è necessario che il database acceda direttamente alla tabella BIBLIOTECA durante l'esecuzione della query. L'operazione INDEX RANGE SCAN dell'indice di chiave primaria è l'unica richiesta per risolvere la query.

La colonna NomeCategoria della tabella BIBLIOTECA possiede un indice non unico sui propri valori. Se nella clausola where della query si specifica una condizione limitativa per i valori di NomeCategoria, viene eseguita un'operazione INDEX RANGE SCAN dell'indice di NomeCategoria. Poiché l'indice BIBLIOTECA\$CATEGORIA non è unico, il database non può eseguire un'operazione INDEX UNIQUE SCAN su BIBLIOTECA\$CATEGORIA, anche se nella query la colonna NomeCategoria viene uguagliata a un unico valore.

## **Quando si usano gli indici**

Dato che gli indici hanno un grande impatto sulle prestazioni delle query, è necessario fare attenzione alle condizioni in cui li si utilizza per risolvere una query. Nei paragrafi che seguono vengono descritte le condizioni che possono richiedere l'utilizzo di un indice nella risoluzione di una query.

### **Se si imposta una colonna indicizzata a un determinato valore**

Nella tabella BIBLIOTECA, la colonna NomeCategoria ha un indice non unico di nome BIBLIOTECA\$CATEGORIA. Una query che mette a confronto la colonna NomeCategoria con un determinato valore sarà in grado di utilizzare l'indice BIBLIOTECA\$CATEGORIA.

La query seguente confronta la colonna NomeCategoria con il valore 'SAGGIADULTI':

```
select Titolo
from BIBLIOTECA
where NomeCategoria = 'SAGGIADULTI';
```

Poiché l'indice BIBLIOTECA\$CATEGORIA non è unico, questa query può restituire più righe e quando si leggono questi dati si potrebbe ricorrere a un'operazione di INDEX RANGE SCAN. In base alle statistiche della tabella, Oracle può decidere di eseguire una scansione completa di tabella.

Se viene utilizzato l'indice, l'esecuzione della query precedente può includere due operazioni: un'operazione INDEX RANGE SCAN di BIBLIOTECA\$CATEGORIA (per ricavare i valori di RowID di tutte le righe contenenti valori "SAGGIADULTI" nella colonna NomeCategoria), seguita da un'operazione TABLE ACCESS BY ROWID della tabella BIBLIOTECA (per recuperare i valori della colonna Titolo).

Se una colonna possiede un indice unico e viene messa a confronto con un valore contenente un segno "=", allora al posto di INDEX RANGE SCAN si userà INDEX UNIQUE SCAN.

### **Se si specifica un intervallo di valori per una colonna indicizzata**

Per utilizzare un indice, non è necessario specificare dei valori espliciti. L'operazione INDEX RANGE SCAN è in grado di analizzare un indice per intervalli di valori. Nell'esempio seguente, viene effettuata una query sulla colonna Titolo della tabella BIBLIOTECA per un certo intervallo di valori (quelli che iniziano con 'M'):

```
select Titolo
from BIBLIOTECA
where Titolo like 'M%';
```

È possibile eseguire una scansione dell'indice anche mentre si utilizzano gli operatori "<" o ">":

```
select Titolo
  from BIBLIOTECA
 where Titolo > 'M';
```

Quando si specifica un intervallo di valori per una colonna, non viene utilizzato un indice per risolvere la query se il primo carattere specificato è un carattere jolly. La query seguente *non* effettuerà un'operazione INDEX RANGE SCAN sugli indici disponibili:

```
select Editore
  from BIBLIOTECA
 where Titolo like '%M%';
```

Dato che il primo carattere della stringa utilizzata per il confronto dei valori è un carattere jolly, l'indice non potrà essere utilizzato per trovare velocemente i dati corrispondenti. In questo caso, viene eseguita una scansione completa della tabella (operazione TABLE ACCESS FULL). In base alle statistiche relative alla tabella e all'indice, Oracle potrebbe scegliere di eseguire una scansione completa dell'indice. In questo esempio, se la colonna selezionata è Titolo, l'ottimizzatore può decidere di eseguire una scansione intera dell'indice di chiave primaria piuttosto che una scansione completa della tabella BIBLIOTECA.

### **Se nessuna funzione viene eseguita sulla colonna nella clausola where**

Si consideri la query seguente che utilizza l'indice BIBLIOTECA\$CATEGORIA:

```
select COUNT(*)
  from BIBLIOTECA
 where NomeCategoria = 'SAGGIADULTI';
```

Come si procede nel caso in cui non si sa se i valori della colonna NomeCategoria sono memorizzati in lettere maiuscole, minuscole, o miste? In una situazione simile, si può scrivere la query in questo modo:

```
select COUNT(*)
  from BIBLIOTECA
 where UPPER(NomeCategoria) = 'SAGGIADULTI';
```

La funzione UPPER cambia i valori di NomeCategoria in lettere maiuscole prima di confrontarli con il valore 'SAGGIADULTI'. Tuttavia, l'uso di questa funzione sulla colonna può impedire all'ottimizzatore di utilizzare un indice sulla colonna stessa. La query precedente (che usa la funzione UPPER) eseguirà un'operazione TABLE ACCESS FULL sulla tabella BIBLIOTECA, a meno che non sia stato creato un indice basato sulla funzione UPPER(NomeCategoria); per i dettagli relativi agli indici basati sulle funzioni, si rimanda al Capitolo 20.

Se si concatenano due colonne o una stringa e una colonna, gli indici su queste colonne non vengono utilizzati. L'indice memorizza il valore reale della colonna e qualsiasi modifica di quel valore impedisce all'ottimizzatore di utilizzarlo.

### **Se nessun controllo di tipo IS NULL o IS NOT NULL viene utilizzato per la colonna indicizzata**

I valori NULL non vengono memorizzati negli indici, quindi la query seguente non utilizza un indice; infatti, in nessun modo l'indice potrebbe contribuire alla risoluzione della query:

```
select Titolo
  from BIBLIOTECA
 where NomeCategoria is null;
```

Dato che NomeCategoria è l'unica colonna con una condizione di limitazione nella query e, in questo caso, la condizione limitativa è un controllo di valore NULL, l'indice BIBLIOTECASCATEGORIA non viene utilizzato e la risoluzione viene effettuata tramite un'operazione TABLE ACCESS FULL.

Che cosa accade se sulla colonna viene effettuato un controllo IS NOT NULL? Tutti i valori non NULL della colonna vengono memorizzati nell'indice; tuttavia la ricerca nell'indice non sarebbe efficiente. Per risolvere la query, l'ottimizzatore dovrebbe leggere ciascun valore dell'indice e accedere alla tabella per ciascuna riga restituita da quest'ultimo. Nella maggior parte dei casi, risulta più efficiente una scansione completa della tabella rispetto a una scansione dell'indice (con le operazioni TABLE ACCESS BY ROWID associate) per tutti i valori restituiti dall'indice. Quindi, la query seguente può non utilizzare un indice:

```
select Titolo
  from BIBLIOTECA
 where NomeCategoria is not null;
```

Se le colonne selezionate si trovano in un indice, l'ottimizzatore può decidere di eseguire una scansione completa di indice al posto di una scansione completa di tabella.

### **Se si utilizzano condizioni di equivalenza**

Negli esempi dei paragrafi precedenti, il valore di Titolo è stato messo a confronto con un valore contenente un segno “=”, come in questa query:

```
select *
  from BIBLIOTECA
 where Titolo = 'INNUMERACY';
```

Cosa accadrebbe se si desiderasse selezionare tutti i record che non hanno un valore di Titolo uguale a 'INNUMERACY'? Il segno = verrebbe sostituito da !=, e la query assumerebbe l'aspetto seguente:

```
select *
  from BIBLIOTECA
 where Titolo != 'INNUMERACY';
```

Quando si risolve questa nuova versione della query, l'ottimizzatore può anche non usare un indice. Gli indici vengono utilizzati quando i valori vengono confrontati esattamente con un altro valore, ossia quando le condizioni di limitazione sono di uguaglianza, non di disuguaglianza. In questo esempio, l'ottimizzatore opterebbe per un indice solo se decidesse che la scansione completa di indice (insieme alle operazioni TABLE ACCESS BY ROWID necessarie per ottenere tutte le colonne) potrebbe essere più rapida di una scansione completa di tabella.

Un altro esempio di disuguaglianza è rappresentato dalla clausola not in, quando impiegata in una subquery. La query seguente seleziona dei valori dalla tabella BIBLIOTECA per libri che non sono stati scritti da Stephen Jay Gould:

```
select *
  from BIBLIOTECA
 where Titolo NOT IN
  (select Titolo
    from BIBLIOTECA_AUTORE
   where NomeAutore = 'STEPHEN JAY GOULD');
```

In alcuni casi, la query del listato precedente non sarebbe in grado di utilizzare un indice sulla colonna Titolo della tabella BIBLIOTECA, poiché non è impostata uguale a nessun valore. Al contrario, il valore di BIBLIOTECA.Titolo viene usato con una clausola not in per eliminare le righe che corrispondono a quelle restituite dalla subquery. Per utilizzare un indice è necessario impostare la colonna indicizzata uguale a un valore. In molti casi, Oracle riscrive internamente la clausola not in trasformandola in una clausola not exists, consentendo così alla query di usare un indice. La query seguente, che usa una clausola in, potrebbe usare un indice sulla colonna BIBLIOTECA.Titolo, oppure potrebbe eseguire un join non indicizzato tra le tabelle; l'ottimizzatore sceglierà il percorso con il minor costo basandosi sulle statistiche disponibili:

```
select *
  from BIBLIOTECA
 where Titolo IN
  (select Titolo
    from BIBLIOTECA_AUTORE
   where NomeAutore = 'STEPHEN JAY GOULD');
```

### **Se la colonna principale di un indice a più colonne è impostata uguale a un valore**

È possibile creare un indice su una sola colonna oppure su più colonne. Nel secondo caso, l'indice viene utilizzato se la sua colonna principale è presente in una condizione di limitazione della query.

Se la query specifica valori solo per le colonne secondarie dell'indice, nelle versioni precedenti a Oracle9i l'indice non veniva utilizzato per risolverla. A partire da Oracle9i, la caratteristica skip-scan dell'indice consente all'ottimizzatore di sfruttare un indice concatenato anche se la sua colonna principale non è elencata nella clausola where.

### **Se si utilizzano le funzioni MAX o MIN**

Se si seleziona il valore MAX o MIN di una colonna indicizzata, l'ottimizzatore può ricorrere all'indice per trovare rapidamente il valore massimo o minimo per la colonna.

### **Se l'indice è selettivo**

Tutte le regole precedenti che stabiliscono se un indice può essere utilizzato considerano la sintassi della query che viene eseguita e la struttura dell'indice disponibile. Se si usa il CBO, l'ottimizzatore può sfruttare la selettività dell'indice per giudicare se un suo eventuale impiego possa ridurre il costo dell'esecuzione della query.

In un indice altamente selettivo, un piccolo numero di record viene associato a ciascun valore distinto della colonna. Per esempio, se in una tabella ci sono 100 record e 80 valori distinti per una colonna di quella tabella, la selettività di un indice su quella colonna è di  $80/100 = 0,80$ . Maggiore è la selettività, minore è il numero di righe restituite per ciascun valore distinto nella colonna.

Il numero di righe restituite per valore distinto è importante nella fase di scansione di un intervallo di valori. Se un indice ha una bassa selettività, le molte operazioni INDEX RANGE SCAN e TABLE ACCESS BY ROWID utilizzate per recuperare i dati possono richiedere un lavoro maggiore rispetto all'operazione TABLE ACCESS FULL di una tabella.

La selettività di un indice non viene considerata dall'ottimizzatore, a meno che non si stia utilizzando il CBO e si sia analizzato l'indice. L'ottimizzatore è in grado di utilizzare istogrammi per esprimere giudizi riguardo alla distribuzione dei dati all'interno di una tabella. Per esempio, se i valori dei dati sono fortemente asimmetrici, tanto che la maggior parte di essi si trova in un intervallo molto ristretto, l'ottimizzatore può evitare di utilizzare l'indice per i valori contenuti in quell'intervallo e sfruttarlo invece per i valori che stanno al di fuori di esso.

## Combinazioni di output da scansioni multiple di un indice

Per risolvere una singola query possono essere utilizzati più indici, oppure scansioni multiple dello stesso indice. Per la tabella BIBLIOTECA sono disponibili due indici: l'indice unico di chiave primaria sulla colonna Titolo, e l'indice BIBLIOTECA\$CATEGORIA sulla colonna NomeCategoria. Nei paragrafi seguenti, viene illustrato in che modo l'ottimizzatore integra l'output di più scansioni attraverso le operazioni AND-EQUAL e INLIST ITERATOR.

### AND-EQUAL di più indici

Se in una query vengono specificate condizioni di limitazione per più colonne indicizzate, l'ottimizzatore può utilizzare più indici quando risolve la query.

La query seguente specifica dei valori sia per la colonna Titolo sia per la colonna NomeCategoria della tabella BIBLIOTECA:

```
select *
  from BIBLIOTECA
 where Titolo > 'M'
   and NomeCategoria > 'B';
```

La clausola where della query contiene due condizioni di limitazione separate. Ciascuna di esse corrisponde a un indice differente: la prima all'indice di chiave primaria, la seconda a BIBLIOTECA\$CATEGORIA. Quando risolve la query, l'ottimizzatore potrebbe decidere di utilizzare entrambi gli indici, oppure di eseguire una scansione completa di tabella.

Se opta per gli indici, ciascuno di essi verrà analizzato tramite un'operazione INDEX RANGE SCAN. I valori di RowID restituiti dalla scansione dell'indice di chiave primaria verranno confrontati con quelli restituiti dalla scansione dell'indice BIBLIOTECA\$CATEGORIA. I valori di RowID restituiti da entrambi gli indici vengono utilizzati durante la successiva operazione TABLE ACCESS BY ROWID. Nella Figura 38.1 è visibile l'ordine di successione delle operazioni.

L'operazione AND-EQUAL, come illustra la Figura 38.1, confronta i risultati delle scansioni dei due indici. In generale, gli accessi a un unico indice a più colonne (in cui la colonna principale viene utilizzata in una condizione di limitazione all'interno della clausola where della query) rispondono meglio rispetto a un'operazione AND-EQUAL di più indici a una sola colonna.

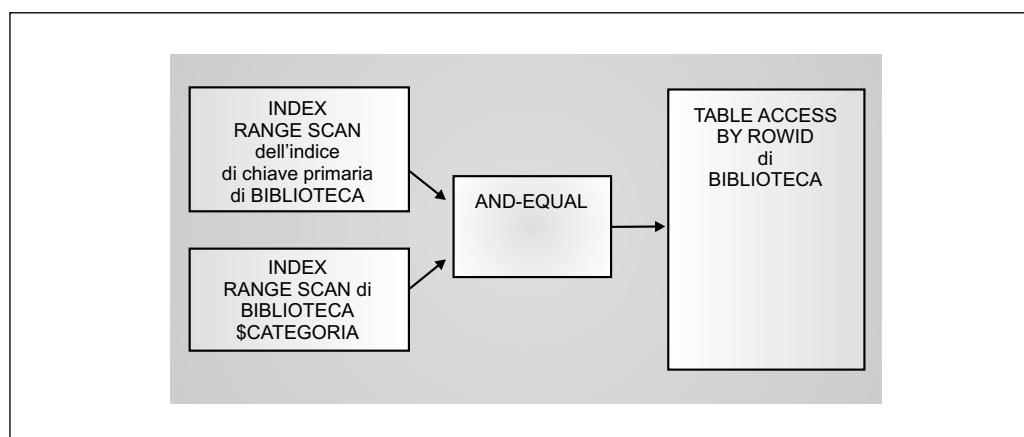


Figura 38.1 Ordine delle operazioni per AND-EQUAL.

## INLIST ITERATION di più scansioni

Se per la condizione di limitazione di una colonna viene specificato un elenco di valori, l'ottimizzatore può eseguire più scansioni e concatenarne i risultati. Per esempio, la query del listato seguente specifica due valori separati per il valore di BIBLIOTECA.NomeCategoria: Un suggerimento INDEX, come descritto nel prossimo paragrafo, consiglia all'ottimizzatore di usare l'indice disponibile al posto di una scansione completa di tabella.

```
select *
  from BIBLIOTECA
 where NomeCategoria in ("SAGGIADULTI", "ILLUSRAGAZZI");
```

Dal momento che non si tratta di un intervallo di valori, un'unica operazione INDEX RANGE SCAN può non essere sufficiente a risolvere la query. Quindi l'ottimizzatore può scegliere di eseguire due scansioni separate dello stesso indice e concatenare i risultati.

Quando risolve la query, l'ottimizzatore può eseguire un'operazione INDEX RANGE SCAN su BIBLIOTECA\$CATEGORIA per ciascuna delle condizioni di limitazione. I valori di RowID restituiti dalle scansioni dell'indice verrebbero utilizzati per accedere alle righe della tabella BIBLIOTECA (per mezzo di operazioni TABLE ACCESS BY ROWID). Le righe restituite da ciascuna delle operazioni TABLE ACCESS BY ROWID possono essere combinate in un unico gruppo di righe attraverso l'operazione CONCATENATION.

In alternativa, l'ottimizzatore potrebbe usare una singola operazione INDEX RANGE SCAN seguita da un'operazione TABLE ACCESS BY ROWID, con un'operazione INLIST ITERATOR che serve per spostarsi tra le righe selezionate della tabella.

## Suggerimenti

Sono disponibili diversi suggerimenti per guidare l'ottimizzatore nella modalità di utilizzo degli indici. INDEX è il suggerimento correlato all'indice più utilizzato. Il suggerimento INDEX comunica all'ottimizzatore di usare una scansione basata sull'indice nella tabella specificata. Con questa opzione non è necessario specificare il nome dell'indice, anche se è possibile elenca-re indici specifici, se lo si desidera.

Per esempio, nella query seguente viene utilizzato il suggerimento INDEX per suggerire l'uso di un indice sulla tabella BIBLIOTECA durante la risoluzione della query:

```
select /*+ index(biblioteca biblioteca$categoria) */ Titolo
  from BIBLIOTECA
 where NomeCategoria = 'SAGGIADULTI';
```

Se si seguono le regole fornite precedentemente in questo paragrafo, questa query dovrebbe utilizzare l'indice senza che vi sia necessità del suggerimento. Tuttavia, se l'indice non è selettivo o la tabella è piccola e si sta utilizzando il CBO, allora l'ottimizzatore potrebbe scegliere di ignorare l'indice. Se si sa che l'indice è selettivo per quanto riguarda i valori dei dati indicati, è possibile specificare il suggerimento INDEX per indurre un percorso di accesso ai dati basato sull'indice.

Nella sintassi del suggerimento, si deve nominare la tabella (oppure il suo alias, se si decide di assegnarne uno alla tabella) e, facoltativamente, inserire il nome dell'indice suggerito. L'ottimizzatore potrebbe anche scegliere di ignorare qualsiasi suggerimento.

Se non si specifica un indice particolare nel suggerimento INDEX, e la tabella in questione possiede più indici, l'ottimizzatore valuta gli indici disponibili e sceglie quello che, con buona

probabilità, presenta un costo minore di scansione. L'ottimizzatore potrebbe inoltre scegliere di eseguire la scansione di più indici e unirli attraverso l'operazione AND-EQUAL descritta nel paragrafo precedente.

Esiste un secondo suggerimento, INDEX\_ASC, che funziona allo stesso modo di INDEX: questa opzione suggerisce una scansione di indice in ordine crescente per risolvere le query di tabelle specifiche. Un terzo suggerimento basato sugli indici, INDEX\_DESC, comunica all'ottimizzatore di eseguire la scansione dell'indice in ordine decrescente (dal valore più alto al valore più basso). Per consigliare una scansione di indice completa e veloce, si utilizzi il suggerimento INDEX\_FFS. Il suggerimento ROWID assomiglia al suggerimento INDEX, che suggerisce l'uso del metodo TABLE ACCESS BY ROWID per la tabella specificata. Il suggerimento AND\_EQUAL suggerisce all'ottimizzatore di unire i risultati di più scansioni di indice.

## **Altri aspetti di messa a punto per gli indici**

Quando si creano degli indici su una tabella, sorgono normalmente due problemi: è opportuno utilizzare più indici o un unico indice concatenato? E se si opta per un indice concatenato, quale colonna impostare come colonna principale dell'indice?

In generale, l'ottimizzatore impiega meno tempo a eseguire la scansione di un unico indice concatenato che non a leggere e unire due indici separati. Quante più righe vengono restituite dalla scansione, tanto più è probabile che la scansione dell'indice concatenato superi come prestazioni l'unione delle scansioni dei due indici. Se si aggiungono ulteriori colonne all'indice concatenato, quest'ultimo risulta meno efficiente per scansioni di intervalli.

Per l'indice concatenato, quale colonna dovrebbe essere quella principale? Questa colonna dovrebbe essere utilizzata di frequente come condizione di limitazione per la tabella e dovrebbe essere altamente selettiva. In un indice concatenato, l'ottimizzatore basa le proprie valutazioni relative alla selettività dell'indice (e quindi alle sue probabilità di essere utilizzato) sulla selettività della colonna principale. Di questi due criteri, il fatto di essere utilizzata nelle condizioni di limitazione e di essere la colonna più selettiva, il primo è il più importante. Se la colonna principale di un indice non è presente in una condizione di limitazione (come descritto in precedenza in questo capitolo), l'indice non viene chiamato in causa, a meno che non si sfrutti la caratteristica skip-scan dell'indice di Oracle9i. Per indurre l'ottimizzatore a usare il metodo skip-scan, potrebbe essere necessario ricorrere a un suggerimento INDEX.

Un indice altamente selettivo basato su una colonna che non è mai presente nelle condizioni di limitazione non viene mai utilizzato. Un indice scarsamente selettivo su una colonna che compare di frequente nelle condizioni di limitazione non incrementa di molto le prestazioni. Se non è possibile creare un indice che sia altamente selettivo e utilizzato di frequente, è opportuno considerare la possibilità di creare indici separati per le colonne da indicizzare.

Molte applicazioni privilegiano i processi di transazione in linea rispetto ai processi batch; possono esserci molti utenti concorrenti in linea ma un numero limitato di utenti concorrenti batch. In generale, le scansioni basate sugli indici consentono agli utenti in linea di accedere ai dati più velocemente rispetto al caso in cui venga eseguita la scansione completa di una tabella. Pertanto, quando si crea un'applicazione è necessario tenere in considerazione i tipi di query eseguite all'interno della stessa e le condizioni di limitazione in queste query. Se si ha una certa familiarità con le query eseguite sul database, si può essere in grado di indicizzare le tabelle in modo che gli utenti in linea possano recuperare velocemente i dati di cui hanno bisogno. Quando le prestazioni del database influiscono direttamente sui processi business in linea, l'applicazione dovrebbe cercare di accedere al database il meno possibile.

## 38.4 Operazioni che gestiscono insiemi di dati

Dopo che i dati sono stati restituiti dalla tabella o dall'indice, è possibile operare su di essi. Si possono raggruppare i record, ordinarli, contarli, bloccarli o unire i risultati di una query con quelli di altre (tramite gli operatori UNION, MINUS e INTERSECT). Nei paragrafi che seguono si imparerà in che modo vengono utilizzate le operazioni che manipolano i dati.

La maggior parte delle operazioni che agiscono su gruppi di record non restituisce i record agli utenti fino a che l'intera operazione non viene completata. Per esempio, l'ordinamento dei record con contemporanea eliminazione dei duplicati (operazione SORT UNIQUE) non può restituire i record all'utente fino a che non si è appurato che tutti i record sono unici. Al contrario, le operazioni di scansione di un indice e quelle di accesso a una tabella sono in grado di restituire un record all'utente appena lo trovano.

Quando viene eseguita un'operazione INDEX RANGE SCAN, la prima riga restituita dalla query passa i criteri delle condizioni di limitazione impostati dalla query stessa, quindi non è necessario valutare il record successivo restituito prima di visualizzare quello già trovato. Al contrario, se viene eseguita un'operazione di gruppo, come un ordinamento, i record non vengono visualizzati immediatamente. Durante operazioni di questo tipo, l'utente deve attendere che tutte le righe vengano elaborate. Quindi, è necessario limitare il numero di operazioni di gruppo effettuate da query utilizzate da utenti in linea (in modo da ridurre il tempo di risposta dell'applicazione). Le operazioni di ordinamento e raggruppamento sono le più comuni nella produzione di report di grandi dimensioni e nelle transazioni batch.

### Ordinamento di righe

Sono tre le operazioni interne di Oracle che ordinano le righe senza raggrupparle. La prima è SORT ORDER BY, utilizzata quando in una query è presente una clausola order by. Per esempio, la query e l'ordinamento della tabella BIBLIOTECA vengono effettuati in base ai valori di Editore:

```
select Titolo from BIBLIOTECA
order by Editore;
```

Quando si esegue questa query, l'ottimizzatore recupera i dati dalla tabella BIBLIOTECA tramite un'operazione TABLE ACCESS FULL (dato che non esistono condizioni di limitazione per la query, tutte le righe vengono restituite). I record recuperati non vengono però visualizzati immediatamente, perché un'operazione SORT ORDER BY ordina i record prima che l'utente possa vedere i risultati.

Occasionalmente, a un'operazione di ordinamento può venire richiesto di eliminare i record doppi mentre li ordina. Per esempio, com'è possibile visualizzare i valori distinti di Editore nella tabella BIBLIOTECA? La query avrebbe l'aspetto seguente:

```
select DISTINCT Editore from BIBLIOTECA;
```

Come per la query precedente, anche questa non ha condizioni di limitazione, per cui viene eseguita un'operazione TABLE ACCESS FULL per recuperare i record dalla tabella BIBLIOTECA. Tuttavia, la funzione DISTINCT indica all'ottimizzatore di restituire solo i valori distinti per la colonna Editore.

Per risolvere la query, l'ottimizzatore prende i record restituiti dall'operazione TABLE ACCESS FULL e li ordina tramite un'operazione SORT UNIQUE. Nessun record viene visualizzato fino a che tutti i record non sono stati elaborati.

Oltre a essere utilizzata dalla funzione DISTINCT, l'operazione SORT UNIQUE viene invocata quando si usano le funzioni MINUS, INTERSECT e UNION (ma non UNION ALL).

Una terza operazione di ordinamento, SORT JOIN, viene sempre utilizzata come parte di un'operazione MERGE JOIN e mai da sola. Le implicazioni di SORT JOIN sulle prestazioni dei join sono descritte nel paragrafo "Operazioni che eseguono join" più avanti in questo capitolo.

## Raggruppamento di righe

Due delle operazioni interne di Oracle ordinano le righe mentre raggruppano assieme i record uguali. Le due operazioni, SORT AGGREGATE e SORT GROUP BY, vengono utilizzate insieme alle funzioni di raggruppamento (come MIN, MAX e COUNT). La sintassi della query determina quale operazione utilizzare.

Nella query seguente viene selezionato dalla tabella BIBLIOTECA il valore massimo di Editore in base all'ordine alfabetico:

```
select MAX(Editore)
      from BIBLIOTECA;
```

Per risolvere la query, l'ottimizzatore esegue due operazioni separate. Per prima cosa un'operazione TABLE ACCESS FULL che seleziona i valori di Editore dalla tabella. In secondo luogo, le righe vengono analizzate tramite un'operazione SORT AGGREGATE, che restituisce all'utente il valore massimo di Editore.

Se la colonna Editore fosse indicizzata, l'indice potrebbe essere utilizzato per risolvere query riguardanti il valore massimo o minimo dell'indice (come descritto in uno dei paragrafi precedenti, "Operazioni che utilizzano gli indici"). Dato che la colonna Editore non è indicizzata, è richiesta un'operazione di ordinamento. Il valore massimo di Editore non viene restituito da questa query fino a che non sono stati letti tutti i record ed è stata completata l'operazione SORT AGGREGATE.

L'operazione SORT AGGREGATE è stata utilizzata nell'esempio precedente perché nella query non compariva alcuna clausola group by. Le query in cui è specificata la clausola group by utilizzano un'operazione interna chiamata SORT GROUP BY.

Come si fa a ricavare il numero dei titoli presenti in ciascun editore? La query seguente seleziona il conteggio di ciascun valore di Editore dalla tabella BIBLIOTECA per mezzo di una clausola group by:

```
select Editore, COUNT(*)
      from BIBLIOTECA
     group by Editore;
```

Questa query restituisce un record per ciascun valore distinto di Editore. Per ogni valore di Editore, il numero delle sue occorrenze nella tabella BIBLIOTECA viene calcolato e visualizzato nella colonna COUNT(\*) .

Per risolvere questa query, Oracle esegue innanzitutto una scansione completa della tabella (non esistono condizioni di limitazione per la query). Dato che si utilizza una clausola group by, le righe restituite dall'operazione TABLE ACCESS FULL vengono elaborate da un'operazione SORT GROUP BY. I record vengono restituiti all'utente dopo che tutte le righe sono state ordinate in gruppi e che il conteggio di ciascun gruppo è stato calcolato. Come con le altre operazioni di ordinamento, nessun record viene restituito all'utente fino a che tutti i record non sono stati elaborati.

Le operazioni incontrate finora hanno coinvolto esempi semplici: scansioni complete di tabelle, scansioni di indici e operazioni di ordinamento. La maggior parte delle query che acce-

dono a un'unica tabella utilizza le operazioni descritte nei paragrafi precedenti. Quando si mette a punto una query per un utente in linea, è consigliabile evitare l'uso di operazioni di ordinamento e di raggruppamento che costringano gli utenti ad attendere che i record vengano elaborati. Quando possibile, è opportuno scrivere query che consentano agli utenti dell'applicazione di ricevere velocemente i record nel momento in cui la query viene risolta. Minore è il numero di operazioni di ordinamento e di raggruppamento effettuate, più rapida sarà la restituzione del primo record trovato all'utente. In una transazione batch, le prestazioni della query vengono misurate in base al tempo complessivo impiegato per suo il completamento, non al tempo necessario per restituire la prima riga. Di conseguenza, le transazioni batch possono utilizzare le operazioni di ordinamento e di raggruppamento senza impatto sulle prestazioni percepibili dell'applicazione.

Se l'applicazione non richiede che tutte le righe vengano ordinate prima della visualizzazione dell'output della query, è opportuno considerare l'eventualità di usare il suggerimento FIRST\_ROWS. FIRST\_ROWS indica all'ottimizzatore di dare la precedenza a percorsi esecutivi che non eseguono operazioni di gruppo.

## Operazioni che utilizzano RowNum

Le query che utilizzano la pseudocolonna RowNum eseguono l'operazione COUNT o COUNT STOPKEY per incrementare il contatore RowNum. Se alla pseudocolonna RowNum viene applicata una condizione di limitazione, del tipo:

```
where RowNum < 10
```

viene utilizzata l'operazione COUNT STOPKEY. Se invece non si specifica alcuna condizione di limitazione per la pseudocolonna RowNum, l'operazione eseguita è COUNT. Le operazioni COUNT e COUNT STOPKEY non hanno relazione con la funzione COUNT.

Mentre viene eseguita, la query seguente utilizza l'operazione COUNT, poiché fa riferimento alla pseudocolonna RowNum:

```
select Titolo,
       RowNum
  from BIBLIOTECA;
```

Per risolvere la query precedente, l'ottimizzatore esegue una scansione completa di indice (sull'indice di chiave primaria della tabella BIBLIOTECA), seguita da un'operazione COUNT per generare i valori di RowNum per ogni riga restituita. L'operazione COUNT non deve attendere che l'intero gruppo di record sia disponibile. Nel momento in cui un record viene restituito dalla tabella BIBLIOTECA, il contatore RowNum viene incrementato e viene determinato il RowNum per quel record.

Nell'esempio seguente viene imposta una condizione di limitazione alla pseudocolonna RowNum:

```
select Titolo,
       RowNum
  from BIBLIOTECA
 where RowNum < 10;
```

Per imporre la condizione di limitazione, l'ottimizzatore sostituisce l'operazione COUNT con COUNT STOPKEY, che mette a confronto il valore incrementato della pseudocolonna

RowNum con la condizione di limitazione applicata. Quando il valore di RowNum supera il valore specificato nella condizione di limitazione, la query non restituisce più alcuna riga.

## UNION, MINUS e INTERSECT

Le funzioni UNION, MINUS e INTERSECT consentono di elaborare e confrontare i risultati di più query. Ciascuna di queste funzioni è associata a un'operazione con lo stesso nome.

La query seguente seleziona tutti i valori di Titolo dalla tabella BIBLIOTECA e dalla tabella LIBRO\_ORDINE:

```
select Titolo
      from BIBLIOTECA
      UNION
select Titolo
      from LIBRO_ORDINE;
```

Quando si effettua la query precedente, l'ottimizzatore esegue ciascuna query separatamente e ne combina i risultati. La prima query è:

```
select Titolo from BIBLIOTECA;
```

Nella query non compaiono condizioni di limitazione, e la colonna Titolo è indicizzata, quindi verrà eseguita la scansione dell'indice di chiave primaria della tabella BIBLIOTECA.

La seconda query è:

```
select Titolo
      from LIBRO_ORDINE
```

Non ci sono condizioni di limitazione nella query, quindi si potrà accedere alla tabella LIBRO\_ORDINE tramite un'operazione TABLE ACCESS FULL.

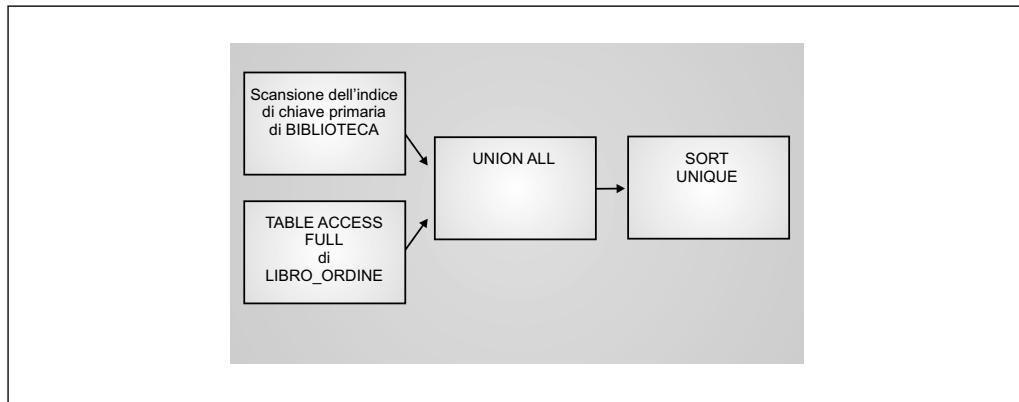
Poiché si esegue una UNION dei risultati delle due query, i due gruppi di risultati vengono uniti tramite un'operazione UNION ALL. Con l'operatore UNION si costringe Oracle a eliminare i record doppi, quindi il gruppo di risultati viene elaborato da un'operazione SORT UNIQUE prima che i record vengano restituiti all'utente. L'operazione dell'ottimizzatore UNION ALL, mostrata nella Figura 38.2, viene utilizzata quando si eseguono entrambe le query union e union all. L'ordine delle operazioni per quanto riguarda l'operatore UNION è visibile nella Figura 38.2.

Se la query avesse utilizzato una funzione UNION ALL al posto di UNION, l'operazione SORT UNIQUE (rappresentata nella Figura 38.2) non sarebbe stata necessaria. In tal caso, la query risulterebbe essere:

```
select Titolo
      from BIBLIOTECA
      UNION ALL
select Titolo
      from LIBRO_ORDINE;
```

Quando si elabora quest'ultima query, l'ottimizzatore esegue le scansioni richieste dalle due query, seguite da un'operazione UNION ALL. Non è richiesta alcuna operazione SORT UNIQUE, poiché una funzione UNION ALL non elimina i record doppi.

Nell'elaborare la query UNION, l'ottimizzatore indirizza separatamente ciascuna delle query destinate alla UNION. Anche se tutti gli esempi presentati nei listati precedenti hanno coinvolto



**Figura 38.2** Ordine delle operazioni per la funzione UNION.

query semplici con scansioni di tabelle complete, le query destinate a UNION possono essere molto articolate, con corrispondenti percorsi di esecuzione complessi. I risultati non vengono restituiti all’utente fino a che tutti i record non sono stati elaborati.

**NOTA** UNION ALL è un’operazione di riga. UNION, che include un’operazione SORT UNIQUE, è un’operazione di gruppo.

Quando si utilizza la funzione MINUS, la query viene elaborata in modo molto simile al percorso esecutivo intrapreso per l’esempio di UNION. Nella query seguente, vengono confrontati i valori di Titolo provenienti dalle tabelle BIBLIOTECA e LIBRO\_ORDINE. Se un valore di Titolo esiste nella tabella LIBRO\_ORDINE ma non in BIBLIOTECA, questo valore verrà restituito dalla query. In altre parole, si desidera visualizzare in ordine tutti i Titoli che non si posseggono ancora.

```

select Titolo
  from LIBRO_ORDINE
 MINUS
select Titolo
  from BIBLIOTECA;

```

Quando si esegue questa query, le due query destinate a MINUS vengono eseguite separatamente. La prima query:

```

select Titolo
  from LIBRO_ORDINE

```

richiede una scansione completa della tabella LIBRO\_ORDINE. La seconda query:

```

select Titolo from BIBLIOTECA;

```

richiede una scansione di indice completa per l’indice di chiave primaria della tabella BIBLIOTECA.

Per eseguire la funzione MINUS, ciascun gruppo di record restituito dalle scansioni complete delle tabelle viene ordinato attraverso un’operazione SORT UNIQUE (con la quale vengono ordinate le righe ed eliminate quelle doppie). L’insieme di righe ordinate viene quindi elaborato

per mezzo dell'operazione MINUS. L'ordine delle operazioni per la funzione MINUS è illustrato nella Figura 38.3.

Come mostrato in questa figura, l'operazione MINUS non viene eseguita fino a che ciascun gruppo di record restituito dalle query non risulta ordinato. Nemmeno l'operazione di ordinamento restituisce i record fino a che non ha terminato di passarli in rassegna tutti, quindi l'operazione MINUS non può iniziare fino a che entrambe le operazioni SORT UNIQUE non sono state completate. Come per l'esempio della query UNION, la query mostrata nell'esempio riguardante l'operazione MINUS offre scarse prestazioni per gli utenti in linea che si basano sulla velocità con cui la query restituisce la prima riga.

La funzione INTERSECT confronta i risultati di due query e stabilisce quali righe sono in comune. La query che segue determina i valori di Titolo che si trovano sia nella tabella BIBLIOTECA sia nella tabella LIBRO\_ORDINE:

```
select Titolo
  from LIBRO_ORDINE
INTERSECT
select Titolo
  from BIBLIOTECA;
```

Per elaborare la query INTERSECT, l'ottimizzatore inizia valutando ciascuna query separatamente. La prima query:

```
select Titolo
  from LIBRO_ORDINE
```

richiede un'operazione TABLE ACCESS FULL sulla tabella LIBRO\_ORDINE. La seconda query:

```
select Titolo from BIBLIOTECA;
```

richiede una scansione intera dell'indice di chiave primaria della tabella BIBLIOTECA. I risultati delle scansioni delle due tabelle vengono elaborati separatamente dalle operazioni SORT UNIQUE. In altre parole, vengono ordinate le righe provenienti dalla tabella LIBRO\_ORDINE, e quelle della tabella BIBLIOTECA. I risultati dei due ordinamenti vengono confrontati dall'operazione INTERSECTION, mentre la funzione INTERSECT restituisce i valori di Titolo restituiti a loro volta da entrambi gli ordinamenti.

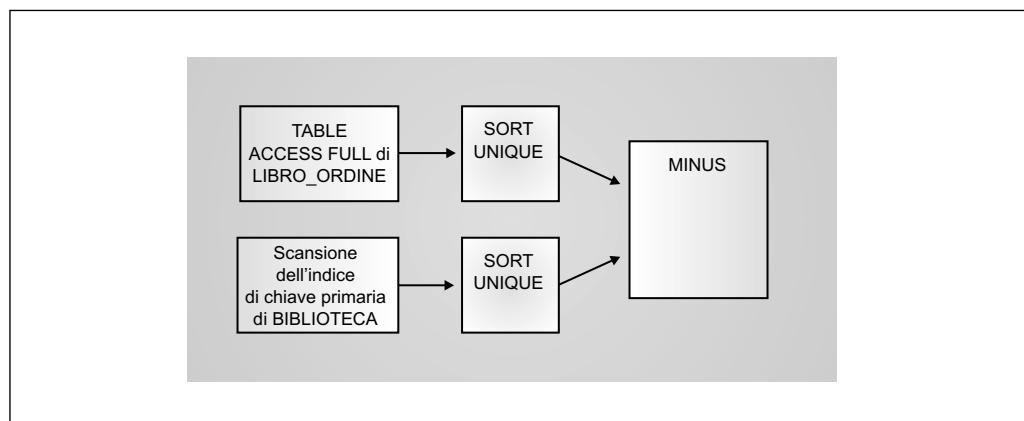


Figura 38.3 Ordine delle operazioni per la funzione MINUS.

La Figura 38.4 illustra l'ordine delle operazioni relativo alla funzione INTERSECT dell'esempio precedente.

Come si può vedere dalla Figura 38.4, il percorso esecutivo di una query che utilizza una funzione INTERSECT richiede l'intervento di operazioni SORT UNIQUE. Dato che le operazioni SORT UNIQUE non restituiscono alcun record all'utente fino a che l'intero gruppo di righe non è stato ordinato, le query che fanno uso della funzione INTERSECT devono attendere che entrambi gli ordinamenti siano completati prima di poter effettuare l'operazione INTERSECTION. A causa di questa dipendenza dalle operazioni di ordinamento, le query che utilizzano la funzione INTERSECT non restituiscono alcun record all'utente fino a che gli ordinamenti non risultano completati.

Tutte le funzioni UNION, MINUS e INTERSECT richiedono l'elaborazione di gruppi di righe prima di restituire qualsiasi riga all'utente. Gli utenti in linea di un'applicazione possono riscontrare cali di prestazioni per le query che utilizzano queste funzioni, anche se gli accessi alle tabelle coinvolte sono stati messi a punto in modo accurato; la dipendenza dalle operazioni di ordinamento influisce sulla velocità con cui la prima riga viene restituita all'utente.

### Selezione di righe per l'aggiornamento

È possibile bloccare le righe utilizzando la sintassi select for update. Per esempio, la query seguente seleziona delle righe dalla tabella LIBRO\_ORDINE e le blocca per evitare che altri utenti se ne impadroniscano. L'uso di select for update consente all'utente di utilizzare la clausola where current of nei comandi insert, update e delete. Un commit invalida il cursore, pertanto è necessario eseguire nuovamente select for update dopo ogni commit.

```
select *
  from LIBRO_ORDINE
    for update of Titolo;
```

Quando si effettua la query precedente, l'ottimizzatore esegue innanzitutto un'operazione TABLE ACCESS FULL per recuperare le righe dalla tabella LIBRO\_ORDINE. L'operazione TABLE ACCESS FULL restituisce le righe appena le trova, senza attendere che l'intero gruppo venga recuperato. Tuttavia, questa query deve eseguire una seconda operazione. L'operazione dell'ottimizzatore FOR UPDATE viene chiamata per bloccare i record. Si tratta di un'operazio-

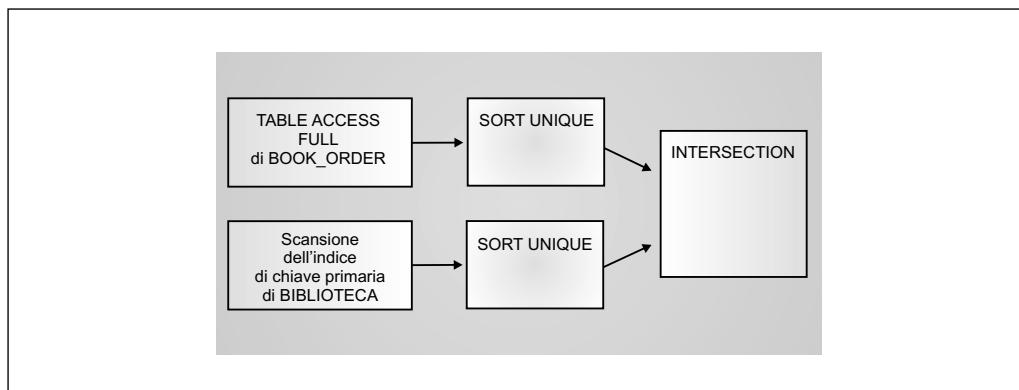


Figura 38.4 Ordine delle operazioni per la funzione INTERSECT.

ne basata su insiemi (analogamente alle operazioni di ordinamento), quindi non restituisce alcuna riga all'utente fino a che l'intero gruppo di righe non è stato bloccato.

## Selezione dalle viste

Quando si crea una vista, Oracle memorizza la query su cui essa si basa. Per esempio, la vista seguente si basa sulla tabella BIBLIOTECA:

```
create or replace view NARRADULTI as
select Titolo, Editore
  from BIBLIOTECA
 where NomeCategoria = 'NARRADULTI';
```

Quando si opera una selezione dalla vista NARRADULTI, l'ottimizzatore preleva i criteri dalla query e li combina con il testo della query della vista. Se nella query della vista si specificano delle condizioni di limitazione, queste se possibile vengono applicate al testo della query della vista. Per esempio, se si esegue la query

```
select Titolo, Editore
  from NARRADULTI
 where Titolo like 'T%';
```

l'ottimizzatore combina le condizioni di limitazione:

```
where Titolo like 'T%';
```

con il testo della query della vista ed esegue la query:

```
select Titolo, Editore
  from BIBLIOTECA
 where NomeCategoria = 'NARRADULTI'
   and Titolo like 'T%';
```

In questo esempio, la vista non ha alcun impatto sulle prestazioni della query. Quando il testo della vista viene unito con le condizioni di limitazione della query, le opzioni che l'ottimizzatore ha a sua disposizione aumentano; potrà scegliere tra più indici e più percorsi di accesso ai dati.

Il modo in cui una vista viene elaborata dipende dalla query su cui si basa. Se il testo della query della vista non può essere unito con la query che utilizza la vista, questa viene risolta prima che il resto delle condizioni venga applicato. Si consideri la vista seguente:

```
create or replace view EDITORE_CONTA as
select Editore, COUNT(*) Conta_Edi
  from BIBLIOTECA
 group by Editore;
```

La vista EDITORE\_CONTA visualizza una riga per ciascun valore distinto di Editore nella tabella BIBLIOTECA insieme al numero dei record che possiedono quel valore. La colonna Conta\_Edi della vista EDITORE\_CONTA registra il conteggio per ciascun valore distinto di Editore.

In che modo l'ottimizzatore elabora la query seguente della vista EDITORE\_CONTA?

---

```
select *
  from EDITORE_CONTA
 where Conta_Edi > 1;
```

La query si riferisce alla colonna Conta\_Edi della vista. Tuttavia, la clausola `where` della query non può essere associata al testo della query della vista, poiché Conta\_Edi viene creata tramite un'operazione di raggruppamento. La clausola `where` potrà essere applicata *dopo* che il gruppo di risultati proveniente dalla vista EDITORE\_CONTA sia stato completamente risolto.

Le viste che contengono operazioni di raggruppamento vengono risolte prima che venga applicata la parte restante dei criteri della query. Analogamente alle operazioni di ordinamento, le viste con operazioni di raggruppamento non restituiscono alcun record fino a che non è stata elaborata l'intera sequenza di risultati. Se la vista non contiene operazioni di raggruppamento, il testo della query può essere unito con le condizioni di limitazione della query che esegue la selezione dalla vista. Di conseguenza, le viste con operazioni di raggruppamento limitano il numero di opzioni disponibili per l'ottimizzatore e non restituiscono i record fino a che tutte le righe sono state elaborate; le query effettuate dagli utenti in linea su queste viste hanno prestazioni scarse.

Quando la query viene elaborata, per prima cosa l'ottimizzatore risolve la vista. Dal momento che la query della vista è:

```
select Editore, COUNT(*) Conta_Edi
  from BIBLIOTECA
 group by Editore;
```

l'ottimizzatore legge i dati dalla tabella BIBLIOTECA tramite un'operazione TABLE ACCESS FULL. Dato che viene utilizzata una clausola `group by`, le righe restituite dall'operazione TABLE ACCESS FULL vengono elaborate da un'operazione SORT GROUP BY. Un'altra operazione, FILTER, elaborerà i dati. L'operazione FILTER serve per eliminare le righe che si basano sui criteri specificati nella query:

```
where Conta_Edi > 1
```

Se si usa una vista che possiede una clausola `group by`, le righe non vengono restituite dalla vista fino a che ciascuna di esse non è stata elaborata dalla vista stessa. Di conseguenza, la query può impiegare parecchio tempo per restituire la prima riga, e le prestazioni della vista percepite dagli utenti in linea possono risultare inaccettabili. Se si riuscissero a rimuovere le operazioni di ordinamento e di raggruppamento dalle viste, si incrementerebbero le probabilità che il testo della vista possa essere unito con quello della query che chiama la vista e di conseguenza le prestazioni potrebbero migliorare (benché la query possa utilizzare altri gruppi di operazioni che influiscono negativamente sulle prestazioni).

## Selezioni da subquery

Ogni volta che è possibile, l'ottimizzatore combina il testo proveniente da una subquery con il resto della query. Per esempio, si consideri la query seguente:

```
select Titolo
  from BIBLIOTECA
 where Titolo in
(select Titolo from LIBRO_ORDINE);
```

L'ottimizzatore, nel valutare la query precedente, determina che la query è funzionalmente equivalente al join seguente delle tabelle BIBLIOTECA e LIBRO\_ORDINE:

```
select BIBLIOTECA.Titolo
  from BIBLIOTECA, LIBRO_ORDINE
 where BIBLIOTECA.Titolo = LIBRO_ORDINE.Titolo;
```

Con la query scritta sotto forma di join, l'ottimizzatore dispone di alcune operazioni per elaborare i dati (come descritto nel paragrafo “Operazioni che eseguono join” più avanti nel capitolo).

Se la subquery non può essere risolta come join, allora viene risolta prima che la parte restante del testo della query venga elaborato rispetto ai dati; si tratta di un meccanismo simile al modo in cui l'operazione FILTER viene utilizzata per le viste. In realtà, l'operazione FILTER viene utilizzata per le subquery se queste non possono essere unite con il resto della query.

Le subquery che dipendono da operazioni di raggruppamento presentano gli stessi problemi di messa a punto delle viste che contengono operazioni di raggruppamento. Le righe recuperate da queste subquery devono essere elaborate completamente prima di poter applicare le restanti condizioni di limitazione della query.

Quando il testo della query viene unito con il testo della vista, aumentano le opzioni a disposizione dell'ottimizzatore. Per esempio, la combinazione delle condizioni di limitazione della query con le condizioni di limitazione della vista può consentire l'uso di un indice prima inutilizzabile durante l'esecuzione della query.

L'unione automatica del testo della query e di quello della vista può essere disattivata tramite il suggerimento NO\_MERGE. Il suggerimento PUSH\_PRED forza un predicato di join in una vista; con PUSH\_SUBQ le subquery non unite vengono valutate il più presto possibile nel percorso esecutivo.

## **Altri aspetti di messa a punto**

Se si effettua la messa a punto di query destinate agli utenti in linea, è opportuno cercare di ridurre il numero di operazioni di ordinamento. Quando si eseguono operazioni che manipolano gruppi di record, è consigliabile ridurre il numero di operazioni di ordinamento annidate.

Per esempio, l'UNION di query in cui ciascuna contiene una clausola group by richiede degli ordinamenti annidati; viene richiesta un'operazione di ordinamento per ciascuna delle query, seguita da un'operazione SORT UNIQUE richiesta per la UNION. L'operazione di ordinamento richiesta per la UNION non può *iniziare* fino a che gli ordinamenti per le clausole group by non sono stati completati. Tanto più profondamente sono annidati gli ordinamenti, tanto maggiore risulta l'impatto delle query sulle prestazioni.

Se si utilizzano funzioni UNION, è opportuno controllare le strutture e i dati delle tabelle per verificare se è possibile che entrambe le query restituiscano gli stessi record. Per esempio, è possibile eseguire la query sui dati da due sorgenti separate e riportare i risultati attraverso un'unica query con la funzione UNION. Se non è possibile che le due query restituiscano le stesse righe, si potrebbe rimpiazzare la funzione UNION con UNION ALL ed evitare così l'operazione SORT UNIQUE eseguita dalla funzione UNION.

Se si utilizza la Parallel Query Option, è opportuno collaborare con il proprio amministratore di database per assicurarsi che il database e le tabelle siano stati configurati in modo adeguato per sfruttare il parallelismo. L'ottimizzatore è in grado di eseguire in parallelo molte operazioni, tra cui le scansioni di tabella, le scansioni di indice e gli ordinamenti. Anche la maggior parte delle operazioni di join, descritte nel prossimo paragrafo, può essere eseguita in parallelo.

### 38.5 Operazioni che eseguono join

Spesso un'unica query ha la necessità di selezionare colonne da più tabelle. A questo scopo, le tabelle vengono unite tramite join nell'istruzione SQL; le tabelle vengono elencate nella clausola from e le condizioni di join vengono elencate nella clausola where. Nell'esempio seguente, vengono unite tramite join le tabelle BIBLIOTECA e LIBRO\_ORDINE in base ai valori comuni della colonna Titolo:

```
select BIBLIOTECA.Titolo
  from BIBLIOTECA, LIBRO_ORDINE
 where BIBLIOTECA.Titolo = LIBRO_ORDINE.Titolo;
```

operazione che può essere riscritta con la sintassi di join introdotta da Oracle9i:

```
select Titolo
  from BIBLIOTECA inner join LIBRO_ORDINE
 using (Titolo);
```

oppure

```
select Titolo
  from BIBLIOTECA natural inner join LIBRO_ORDINE;
```

Le condizioni di join possono agire come condizioni di limitazione per il join stesso. Dato che la colonna BIBLIOTECA.Titolo è uguagliata a un valore nella clausola where, l'ottimizzatore può, durante l'esecuzione della query, utilizzare un indice sulla colonna BIBLIOTECA.Titolo. Se la colonna LIBRO\_ORDINE.Titolo dispone di un indice, anche questo indice può essere preso in considerazione dall'ottimizzatore.

Oracle presenta tre metodi per elaborare i join: le operazioni MERGE JOIN, le operazioni NESTED LOOPS e le operazioni HASH JOIN. In base alle condizioni presenti nella query, agli indici disponibili e alle statistiche disponibili (per il CBO), l'ottimizzatore sceglie quale operazione di join utilizzare. In relazione alla natura dell'applicazione e delle query, si può decidere di indurre l'ottimizzatore a utilizzare un metodo differente dalla sua prima scelta. Nei paragrafi che seguono vengono analizzate le caratteristiche dei diversi metodi di join e le condizioni in cui ciascuno di essi si rivela più utile.

### Gestione dei join di più di due tabelle da parte di Oracle

Se in una query vengono unite tramite join più di due tabelle, l'ottimizzatore tratta la query come un insieme di più join. Per esempio, se la query unisce tramite join tre tabelle, l'ottimizzatore unisce prima due tabelle, quindi i risultati così ottenuti con la terza tabella. La dimensione dei risultati del join iniziale influenza sulle prestazioni dei join restanti. Se la dimensione dei risultati ottenuti con il primo join è di una certa entità, il secondo join dovrà elaborare molte righe.

Se nella query vengono unite tramite join tre tabelle di dimensione variabile, per esempio una tabella piccola di nome PICCOLA, una tabella di medie dimensioni di nome MEDIA e una più grande chiamata GRANDE, è necessario prestare attenzione all'ordine in cui le tabelle vengono unite. Se il join di MEDIA con GRANDE restituisce molte righe, il join del risultato così ottenuto con la tabella PICCOLA può richiedere parecchio lavoro. In alternativa, se vengono unite tramite join per prime le tabelle PICCOLA e MEDIA, il join del risultato e della tabella GRANDE può ridurre la quantità di lavoro eseguito dalla query. Più avanti nel capitolo il para-

grafo “Visualizzazione dei percorsi di esecuzione”, che descrive i comandi `explain plan` e `set autotrace on`, mostra in che modo si può interpretare l’ordine dei join.

## MERGE JOIN

In questa operazione, i due input del join vengono elaborati separatamente, ordinati e uniti tramite join. Le operazioni MERGE JOIN vengono comunemente utilizzate quando non vi sono indici disponibili per le condizioni di limitazione della query.

Nella query seguente vengono unite tramite join le tabelle BIBLIOTECA e BIBLIOTECA\_AUTORE. Se nessuna delle tabelle possiede un indice sulla colonna Titolo, non ci sono indici che possano essere utilizzati durante la query (poiché esistono altre condizioni di limitazione nella query). Questo esempio ricorre a un suggerimento per indurre l’uso di un’operazione merge join:

```
select /*+ USE_MERGE (biblioteca, biblioteca_autore)*/
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA_AUTORE
 where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo
   and BIBLIOTECA.Editore > 'T%';
```

oppure:

```
select /*+ USE_MERGE (biblioteca, biblioteca_autore)*/
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA inner join BIBLIOTECA_AUTORE
    on BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo
 where BIBLIOTECA.Editore > 'T%';
```

Per risolvere la query, l’ottimizzatore può scegliere di eseguire un’operazione MERGE JOIN delle tabelle. Questa operazione richiede che ciascuna delle tabelle venga letta separatamente (in genere tramite un’operazione TABLE ACCESS FULL oppure una scansione di indice completa). Il gruppo di righe restituito dalla scansione della tabella BIBLIOTECA\_AUTORE viene ordinato tramite un’operazione SORT JOIN. Il gruppo di righe restituito dalla scansione di indice e dall’operazione TABLE ACCESS BY ROWID della tabella BIBLIOTECA è già ordinato (nell’indice), quindi per questa parte del percorso esecutivo non è necessaria un’altra operazione SORT JOIN. I dati ricavati dalle operazioni SORT JOIN vengono uniti con un’operazione MERGE JOIN. Nella Figura 38.5 è possibile vedere l’ordine delle operazioni per la MERGE JOIN dell’esempio. L’explain plan è il seguente:

Execution Plan

```
-----
0      SELECT STATEMENT Optimizer=CHOOSE (Cost=7 Card=5 Byte=320)
1      0      MERGE JOIN (Cost=7 Card=5 Byte=320)
2      1      TABLE ACCESS (BY INDEX ROWID) OF 'BIBLIOTECA' (Cost=2 Car
            d=4 Byte=120)
3      2          INDEX (FULL SCAN) OF 'SYS_C002547' (UNIQUE) (Cost=1 Ca
            rd=4)
4      1      SORT (JOIN) (Cost=5 Card=37 Byte=1258)
5      4      TABLE ACCESS (FULL) OF 'BIBLIOTECA_AUTORE' (Cost=1 Card
            =37 Byte=1258)
```

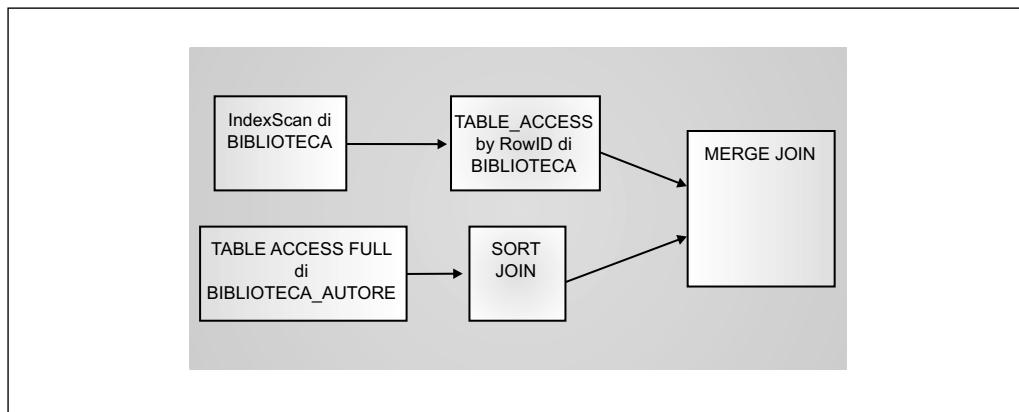


Figura 38.5 Ordine delle operazioni per MERGE JOIN.

Quando si utilizza un’operazione MERGE JOIN per unire tramite join due gruppi di record, ciascun gruppo viene elaborato separatamente prima di essere unito. L’operazione MERGE JOIN non può iniziare fino a che non ha ricevuto i dati da entrambe le operazioni SORT JOIN che le forniscono l’input. Le operazioni SORT JOIN, a loro volta, non forniscono dati a MERGE JOIN fino a che tutte le righe non sono state ordinate. Se gli indici fungono da sorgenti di dati, le operazioni SORT JOIN possono essere ignorate.

Se l’operazione MERGE JOIN deve attendere il completamento di due operazioni SORT JOIN separate, un join che utilizza MERGE JOIN normalmente offre prestazioni scarse per gli utenti in linea. Questo problema è dovuto al ritardo nella restituzione della prima riga del join agli utenti. Se le tabelle aumentano di dimensione, il tempo richiesto per il completamento degli ordinamenti aumenta drasticamente. Se le tabelle hanno dimensioni molto diverse tra di loro, l’ordinamento eseguito sulla tabella più grande influisce negativamente sulle prestazioni complessive della query.

Dal momento che le operazioni MERGE JOIN richiedono una scansione e un ordinamento completi delle tabelle coinvolte, queste operazioni dovrebbero essere effettuate solo se entrambe le tabelle sono molto piccole o molto grandi. Se entrambe le tabelle sono molto piccole, il processo di scansione e ordinamento viene completato velocemente. Se entrambe le tabelle sono molto grandi, le operazioni di ordinamento e scansione richieste dalle operazioni MERGE JOIN possono sfruttare le opzioni parallele di Oracle.

Oracle è in grado di eseguire operazioni in parallelo, permettendo così a più processori di partecipare all’esecuzione di un unico comando. Tra le operazioni che possono essere eseguite in parallelo vi sono quelle TABLE ACCESS FULL e le operazioni di ordinamento. Dato che MERGE JOIN le utilizza entrambe, può ottenere grandi vantaggi dalle opzioni parallele di Oracle. L’esecuzione in parallelo di query che coinvolgono operazioni MERGE JOIN spesso incrementa le prestazioni (purché vi siano adeguate risorse di sistema disponibili per supportare le operazioni parallele).

## NESTED LOOPS

Le operazioni NESTED LOOPS consentono di unire tramite join due tabelle con un metodo di risoluzione ciclica: i record vengono recuperati da una tabella e, per ciascun record recuperato, viene effettuato un accesso alla seconda tabella, che avviene in base a un indice.

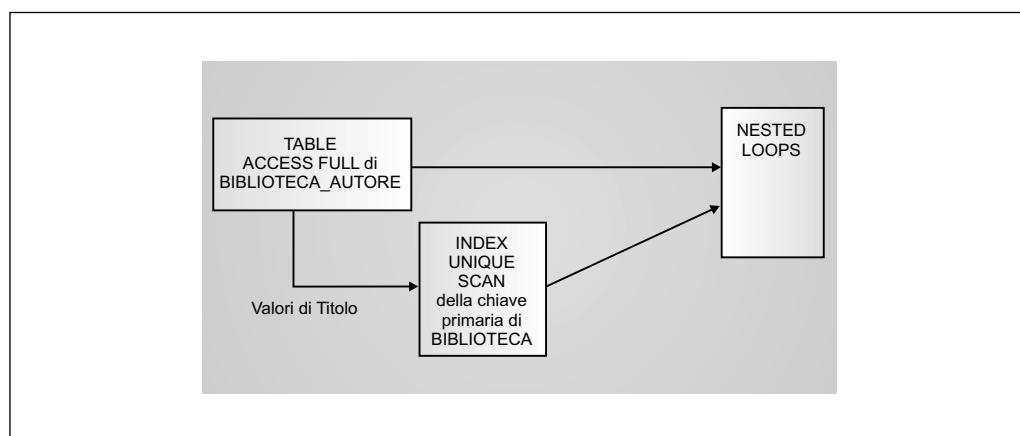
Nel listato seguente, viene riproposta una forma della query del paragrafo precedente MERGE JOIN. Viene utilizzato un suggerimento per consigliare un accesso basato sugli indici per la tabella BIBLIOTECA:

```
select /*+ INDEX(biblioteca) */
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA_AUTORE
 where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo;
```

Dal momento che la colonna Titolo della tabella BIBLIOTECA viene utilizzata come parte della condizione di join nella query, l'indice di chiave primaria è in grado di risolvere il join. Quando si esegue la query, per effettuare il join può essere utilizzata un'operazione NESTED LOOPS.

Per effettuare un join NESTED LOOPS, per prima cosa l'ottimizzatore deve selezionare una *tabella guida* per il join, ovvero la tabella che viene letta per prima (solitamente attraverso un'operazione TABLE ACCESS FULL, anche se accade molto spesso di vedere scansioni di indice). Per ciascun record nella tabella guida verrà eseguita una query sulla seconda tabella all'interno del join. La query di esempio unisce join le tabelle BIBLIOTECA e BIBLIOTECA\_AUTORE in base ai valori della colonna Titolo. Durante l'esecuzione di NESTED LOOPS, un'operazione selezionerà tutti i record dalla tabella BIBLIOTECA\_AUTORE. Viene controllato l'indice di chiave primaria della tabella BIBLIOTECA per stabilire se contiene una voce per il valore nel record corrente prelevato dalla tabella BIBLIOTECA\_AUTORE. Se si trova una corrispondenza, allora il valore di Titolo verrà restituito dall'indice di chiave primaria di BIBLIOTECA. Qualora fosse necessario prelevare altre colonne da BIBLIOTECA, la riga verrebbe selezionata dalla tabella BIBLIOTECA tramite un'operazione TABLE ACCESS BY ROWID. La Figura 38.6 mostra il flusso delle operazioni per il join NESTED LOOPS.

Come si può vedere dalla Figura 38.6, in un join NESTED LOOPS vengono coinvolte almeno due operazioni di accesso ai dati: un accesso alla tabella guida e un accesso, solitamente basato sugli indici, alla tabella dipendente. I metodi di accesso ai dati più utilizzati, TABLE ACCESS FULL, TABLE ACCESS BY ROWID e le scansioni di indice, restituiscono dei record alle operazioni successive non appena ne trovano uno; non attendono che l'intero gruppo di record venga selezionato. Dato che queste operazioni sono in grado di fornire rapidamente agli



**Figura 38.6** Ordine delle operazioni per NESTED LOOPS.

utenti le prime righe corrispondenti, i join NESTED LOOPS vengono riservati normalmente per gli utenti in linea.

Nell'implementazione di join NESTED LOOPS, è necessario considerare la dimensione delle tabella guida. Se la tabella guida è grande e viene letta tramite una scansione completa di tabella, l'operazione TABLE ACCESS FULL eseguita su di essa può influire negativamente sulle prestazioni della query. Se gli indici sono disponibili su entrambi i lati del join, Oracle selezionerà una tabella guida per la query. Il metodo della selezione per tabelle guida dipende dall'ottimizzatore utilizzato. Se si usa il CBO, l'ottimizzatore effettua un controllo per quanto riguarda la dimensione delle tabelle e la selettività degli indici, scegliendo il percorso con il costo complessivo minore. Se invece si usa RBO e sono disponibili degli indici per ciascuna delle condizioni di join, la tabella guida risulta solitamente essere la tabella elencata per *ultima* nella clausola from.

Quando si uniscono tramite join tre tabelle, Oracle esegue due join separati: un join di due tabelle che genera un insieme di record e successivamente un join tra questo insieme di record e la terza tabella. Se si utilizzano join NESTED LOOPS, l'ordine in cui le tabelle vengono unite si complica. L'output del primo join genera un insieme di record che funge da tabella guida per il secondo join.

La dimensione del gruppo di record restituiti dal primo join influisce sulle prestazioni del secondo join, pertanto può avere un impatto significativo sulle prestazioni dell'intera query. Si dovrebbe provare per prima cosa a unire tramite join le tabelle più selettive, in modo che l'impatto esercitato da questi join su quelli futuri sia trascurabile. Se nel primo join di una query a più join vengono assemblate tabelle di grandi dimensioni, la dimensione di queste tabelle influirà su ciascun join successivo e avrà un impatto negativo sulle prestazioni complessive della query.

I join NESTED LOOPS risultano utili quando le tabelle coinvolte nel join hanno dimensioni diverse; infatti è possibile utilizzare la tabella più piccola come tabella guida ed effettuare selezioni dalla tabella più grande tramite un accesso basato su indice. Tanto più l'indice è selettivo, tanto più velocemente viene completata la query.

## HASH JOIN

L'ottimizzatore può decidere in modo dinamico di eseguire dei join con l'operazione HASH JOIN al posto di MERGE JOIN o NESTED LOOPS. Questa operazione confronta due tabelle in memoria. Durante un'operazione HASH JOIN viene eseguita la scansione della prima tabella e il database applica le funzioni di "hash" ai dati per preparare la tabella al join. I valori derivati dalla seconda tabella vengono letti (solitamente tramite un'operazione TABLE ACCESS FULL) e viene utilizzata la funzione di hash per confrontare la seconda tabella con la prima. Le righe corrispondenti vengono restituite all'utente.

**NOTA** *Benché abbiano nomi simili, gli hash join non hanno nulla a che vedere con i cluster hash o con le operazioni TABLE ACCESS HASH trattate più avanti in questo capitolo.*

L'ottimizzatore può scegliere di eseguire hash join anche se gli indici sono disponibili. Nella query d'esempio del listato seguente, le tabelle BIBLIOTECA e BIBLIOTECA\_AUTORE vengono unite tramite join sulla colonna Titolo:

```
select /*+ USE_HASH (biblioteca) */
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA_AUTORE
 where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo;
```

La tabella BIBLIOTECA ha un indice unico sulla colonna Titolo. Dato che l'indice è disponibile e può essere utilizzato per valutare le condizioni di join, l'ottimizzatore può scegliere di eseguire un join NESTED LOOPS delle due tabelle, come mostrato nel paragrafo precedente. Se si esegue un hash join, ciascuna tabella viene letta tramite un'operazione separata. I dati ricavati dalle scansioni della tabella e dell'indice fungono da input per un'operazione HASH JOIN.

Nella Figura 38.7 è possibile vedere l'ordine delle operazioni per un hash join. Come si può vedere dalla figura, l'hash join non si basa su operazioni che elaborano gruppi di righe. Le operazioni coinvolte in hash join restituiscono velocemente i record agli utenti. Gli hash join sono adatti per le query eseguite da utenti in linea se le tabelle sono piccole e la loro scansione può essere completata in breve tempo.

## Elaborazione di outer join

Nell'elaborare un outer join, l'ottimizzatore utilizza uno dei tre metodi descritti nei paragrafi precedenti. Per esempio, se una query di esempio eseguisse un outer join tra le tabelle LIBRO\_ORDINE e CATEGORIA (per vedere quali categorie non hanno libri in ordine), al posto di un'operazione NESTED LOOPS si potrebbe optare per NESTED LOOPS OUTER. In un'operazione NESTED LOOPS OUTER, la tabella esterna per l'outer join viene normalmente utilizzata come tabella guida per la query; mentre viene eseguita la scansione dei record della tabella interna alla ricerca di record corrispondenti, vengono restituiti valori NULL per le righe che non trovano corrispondenza.

## Suggerimenti

I suggerimenti possono servire per ignorare le selezioni dell'ottimizzatore relative a un metodo di join. Con i suggerimenti è possibile specificare quale metodo di join utilizzare oppure l'obiettivo di quest'ultimo.

Oltre ai suggerimenti descritti in questo paragrafo, si possono utilizzare le opzioni FULL e INDEX descritte in precedenza per influire sul modo in cui vengono elaborati i join. Per esempio, se si utilizza un suggerimento per indurre l'esecuzione di un join NESTED LOOPS, si potrebbe

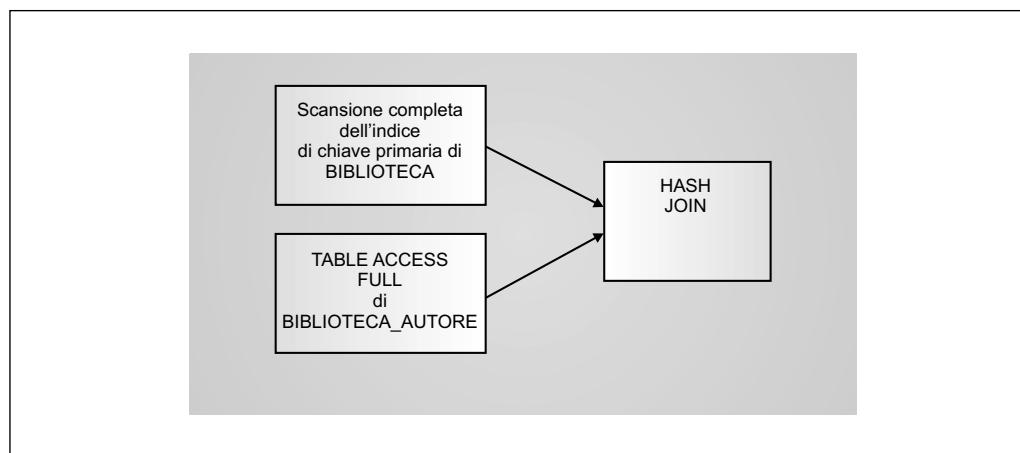


Figura 38.7 Ordine delle operazioni per HASH JOIN.

usare anche un suggerimento INDEX per specificare quale indice utilizzare per il join e a quale tabella accedere attraverso una scansione completa.

### Suggerimenti relativi agli obiettivi

È possibile specificare un suggerimento che indirizzi l'ottimizzatore a eseguire una query con un obiettivo specifico. Gli obiettivi disponibili, per quanto concerne i join, sono i seguenti:

- Esegue le query in modo da restituire tutte le righe il più velocemente possibile.
- Esegue la query in modo da restituire la prima riga il più velocemente possibile.

Per default, l'ottimizzatore esegue una query con un percorso d'esecuzione che consenta di ridurre il tempo globale necessario per risolverla. Quindi l'impostazione predefinita prevede l'uso di ALL\_ROWS come obiettivo. Se l'ottimizzatore è interessato solamente al tempo totale necessario per restituire tutte le righe per la query, possono essere utilizzate le operazioni basate su insiemi come gli ordinamenti e le operazioni MERGE JOIN. Tuttavia, l'obiettivo ALL\_ROWS non è sempre appropriato. Per esempio, gli utenti in linea tendono a giudicare le prestazioni di un sistema in base al tempo che la query richiede per restituire la prima riga di dati. In questo modo, gli utenti hanno come obiettivo primario l'opzione FIRST\_ROWS e come obiettivo secondario il tempo richiesto per restituire tutte le righe.

I suggerimenti disponibili consentono di mimetizzare gli obiettivi: il suggerimento ALL\_ROWS consente all'ottimizzatore di operare una selezione tra tutte le operazioni disponibili per ridurre al minimo il tempo complessivo di elaborazione per la query, mentre il suggerimento FIRST\_ROWS indica all'ottimizzatore di selezionare un percorso esecutivo che riduca il tempo necessario per restituire all'utente la prima riga trovata.

**NOTA** È opportuno utilizzare i suggerimenti ALL\_ROWS e FIRST\_ROWS solo se prima sono state analizzate le tabelle.

Nell'esempio seguente, viene modificata la query ricavata dagli esempi di join in modo da includere il suggerimento ALL\_ROWS:

```
select /*+ ALL_ROWS */
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA_AUTORE
 where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo;
```

La stessa query può essere modificata in modo da utilizzare FIRST\_ROWS:

```
select /*+ FIRST_ROWS */
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA_AUTORE
 where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo;
```

Con il suggerimento FIRST\_ROWS, è più probabile che l'ottimizzatore utilizzi NESTED LOOPS che non MERGE JOIN. Il metodo di join selezionato dipende in parte dal resto della query. Per esempio, la query di join non contiene una clausola order by (che è un'operazione di insieme eseguita dall'operazione SORT ORDER BY). Se la query viene corretta in modo da contenere una clausola order by, come nel listato seguente, in che modo il processo di join ne risulta influenzato?

```
select /*+ FIRST_ROWS */
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA_AUTORE
```

---

```
where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo
order by BIBLIOTECA_AUTORE.NomeAutore;
```

Con l'aggiunta di una clausola `order by` alla query, `SORT ORDER BY` risulta l'ultima operazione eseguita prima di mostrare l'output all'utente. L'operazione `SORT ORDER BY` non viene completata, e l'utente non vede alcun record, fino a che tutti i record non sono stati ordinati. Quindi in questo esempio il suggerimento `FIRST_ROWS` indica all'ottimizzatore di eseguire il join il più velocemente possibile, fornendo al più presto i dati all'operazione `SORT ORDER BY`. L'aggiunta dell'operazione di ordinamento (la clausola `order by`) nella query può annullare o modificare l'impatto del suggerimento `FIRST_ROWS` sul percorso esecutivo della query (in quanto un'operazione `SORT ORDER BY` è lenta nel restituire i record all'utente, a prescindere dal metodo di join scelto).

### **Suggerimenti riguardanti i metodi**

Oltre a specificare gli obiettivi che l'ottimizzatore deve avere per valutare le alternative dei metodi di join, è possibile elencare le operazioni specifiche da eseguire e le tabelle sulle quali andranno a operare. Se una query coinvolge solo due tabelle, quando si fornisce un suggerimento per il metodo da utilizzare, non è necessario specificare quali tabelle unire tramite join.

Il suggerimento `USE_NL` indica all'ottimizzatore di utilizzare l'operazione `NESTED LOOPS` per unire tramite join le tabelle. Nell'esempio seguente, viene specificato il suggerimento `USE_NL` per l'esempio relativo alla query di join. All'interno del suggerimento, la tabella `BIBLIOTECA` viene specificata come tabella interna per il join.

```
select /*+ USE_NL(biblioteca) */
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA_AUTORE
 where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo;
```

Se si desidera che tutti i join di una query su più tabelle utilizzino operazioni `NESTED LOOPS`, è sufficiente specificare il suggerimento `USE_NL` senza nessun riferimento alle tabelle. In generale, è opportuno indicare i nomi delle tabelle ogni volta che si utilizza un suggerimento per specificare un metodo di join, perché non si sa in che modo la query verrà utilizzata in futuro. Persino la configurazione attuale degli oggetti del database potrebbe non essere nota; per esempio, uno degli oggetti nella clausola `from` potrebbe essere una vista che è stata messa a punto per utilizzare le operazioni `MERGE JOIN`. Se in un suggerimento si specifica una tabella, è necessario far riferimento all'alias della tabella oppure al nome non qualificato della tabella stessa. Ovvero, se la clausola `from` fa riferimento a una tabella come:

```
from GIO.SUA_TABELLA, IO.MIA_TABELLA
```

*non* va specificato un suggerimento del tipo:

```
/*+ USE_NL(IO.MIA_TABELLA) */
```

Al contrario, è necessario fare riferimento alla tabella mediante il suo nome, senza il proprietario:

```
/*+ USE_NL(mia_tabella) */
```

Se esistono più tabelle con lo stesso nome, occorre assegnare loro degli alias e farvi riferimento nel suggerimento. Per esempio, nel caso in cui una tabella venga unita tramite join a se stessa, la clausola `from` potrebbe includere il testo mostrato di seguito:

---

from BIBLIOTECA S1, BIBLIOTECA S2

Nel listato seguente è possibile vedere un suggerimento che forza il join BIBLIOTECA-BIBLIOTECA a utilizzare NESTED LOOPS, nel quale viene specificato l'alias della tabella:

```
/*+ USE_NL(s2) */
```

L'ottimizzatore ignora i suggerimenti scritti con una sintassi non adatta. Qualsiasi suggerimento scritto con una sintassi inadatta viene considerato come un commento (in quanto racchiuso tra la sequenza di caratteri /\* e \*/).

**NOTA** *USE\_NL è un suggerimento, non una regola. L'ottimizzatore lo può riconoscere e scegliere comunque di ignorarlo, in base alle statistiche disponibili al momento dell'esecuzione della query.*

Se si utilizzano i join NESTED LOOPS, è necessario prestare attenzione all'ordine in cui le tabelle vengono unite tramite join. Il suggerimento ORDERED, quando utilizzato con i join NESTED LOOPS, influenza sull'ordine con cui vengono unite le tabelle.

Quando si specifica il suggerimento ORDERED, le tabelle vengono unite nell'ordine in cui sono elencate nella clausola from della query. Se la clausola from contiene tre tabelle, come:

```
from LIBRO_ORDINE, BIBLIOTECA, BIBLIOTECA_AUTORE
```

le prime due tabelle vengono unite tramite il primo join e il risultato di quest'ultimo viene unito alla terza tabella.

Dal momento che l'ordine dei join è fondamentale per le prestazioni dei join NESTED LOOPS, il suggerimento ORDERED viene spesso utilizzato in abbinamento a USE\_NL. Se si specificano dei suggerimenti per stabilire l'ordine dei join, è necessario assicurarsi che la distribuzione dei valori all'interno delle tabelle da unire non cambi eccessivamente nel tempo; in caso contrario l'ordine specificato potrebbe in futuro incidere negativamente sulle prestazioni.

È possibile utilizzare USE\_MERGE per suggerire all'ottimizzatore di eseguire un'operazione MERGE JOIN tra le tabelle specificate. Nel listato seguente, il suggerimento comunica all'ottimizzatore di effettuare un'operazione MERGE JOIN tra le tabelle BIBLIOTECA e BIBLIOTECA\_AUTORE:

```
select /*+ USE_MERGE (biblioteca, biblioteca_autore)*/
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA_AUTORE
 where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo
   and BIBLIOTECA.Editore > 'T%';
```

Con il suggerimento USE\_HASH è possibile comunicare all'ottimizzatore di considerare l'eventualità di utilizzare un metodo HASH JOIN. Se non viene specificata alcuna tabella, l'ottimizzatore seleziona la prima tabella per la quale eseguire una scansione nella memoria basandosi sulle statistiche disponibili.

```
select /*+ USE_HASH (biblioteca) */
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA_AUTORE
 where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo;
```

Se si usa il CBO, ma precedentemente le query sono state messe a punto per sfruttare l'ottimizzazione basata su regole, è possibile comunicare al CBO di utilizzare il metodo basato

sulle regole per elaborare la query. Il suggerimento RULE indica all'ottimizzatore di utilizzare RBO per ottimizzare la query; tutti gli altri suggerimenti all'interno della query vengono ignorati. Nell'esempio seguente, il suggerimento RULE viene utilizzato durante un join:

```
select /*+ RULE */
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA_AUTORE
 where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo;
```

In generale, si dovrebbe utilizzare il suggerimento RULE solo se le query sono state messe a punto specificamente per RBO. Nonostante il suggerimento RULE venga tuttora gestito, è consigliabile considerare in sua vece l'uso del CBO per le query.

È possibile impostare l'obiettivo dell'ottimizzatore a livello di sessione tramite il comando `alter session`. Nell'esempio seguente, il parametro `optimizer_goal` della sessione è stato modificato in RULE:

```
alter session set optimizer_goal = RULE
```

Le altre impostazioni del parametro `optimizer_goal` della sessione includono COST, CHOOSE, ALL\_ROWS e FIRST\_ROWS.

Tra gli altri suggerimenti che influiscono sui join troviamo LEADING (per indicare a Oracle di usare la tabella specificata per prima nell'ordine di join) e ORDERED (per unire tramite join le tabelle seguendo l'ordine in cui sono elencate nella clausola `from`).

## Altri aspetti della messa a punto

Come si è già evidenziato nella trattazione delle operazioni NESTED LOOPS e MERGE JOIN, le operazioni differiscono per il tempo che impiegano a restituire la prima riga dalla query. Dato che MERGE JOIN si basa su operazioni di insieme, non restituisce alcun record all'utente fino a che tutte le righe non sono state elaborate. D'altra parte, NESTED LOOPS è in grado di restituire le righe all'utente appena risultano disponibili.

Dal momento che i join NESTED LOOPS sono in grado di restituire velocemente le righe agli utenti, vengono impiegati spesso per le query eseguite di frequente dagli utenti in linea. La loro efficienza nel restituire la prima riga, tuttavia, viene spesso influenzata da operazioni basate su insiemi applicate alle righe che sono state selezionate. Per esempio, l'aggiunta di una clausola `order by` a una query comporta un'operazione SORT ORDER BY alla fine dell'elaborazione della query e nessuna riga viene visualizzata dall'utente fino a che tutte le righe non sono state ordinate.

Come descritto nel paragrafo “Operazioni che utilizzano gli indici”, più indietro nel capitolo, l'uso delle funzioni su una colonna impedisce al database di utilizzare un indice su questa colonna durante le ricerche di dati, a meno che non sia stato creato un indice basato sulle funzioni. Queste informazioni possono servire per disattivare in modo dinamico gli indici e influire sul metodo di join scelto. Per esempio, la query seguente non specifica un suggerimento di join, ma disattiva l'uso degli indici sulla colonna Titolo concatenandone i valori con una stringa nulla:

```
select BIBLIOTECA_AUTORE.NomeAutore
      from BIBLIOTECA, BIBLIOTECA_AUTORE
     where BIBLIOTECA.Titolo||'' = BIBLIOTECA_AUTORE.Titolo||'';
```

La disattivazione dinamica degli indici consente di forzare l'utilizzo di operazioni MERGE JOIN anche se è attivo RBO (che non accetta alcun suggerimento).

Come si è già osservato durante la trattazione dell’operazione NESTED LOOPS, l’ordine dei join è importante quanto il metodo selezionato. Se un join di grandi dimensioni o non selettivo è il primo di una serie, la grande mole di dati restituiti influirà negativamente sulle prestazioni dei join successivi della query, così come sulle prestazioni della query nel suo complesso.

In base ai suggerimenti, all’obiettivo dell’ottimizzatore e alle informazioni statistiche, l’ottimizzatore può scegliere di utilizzare un’ampia varietà di metodi di join all’interno della stessa query. Per esempio, l’ottimizzatore può valutare una query di tre tabelle come un join NESTED LOOPS di due tabelle, seguito da un MERGE JOIN dell’output prodotto dall’operazione NESTED LOOPS con la terza tabella. Combinazioni simili di tipi di join si trovano solitamente quando è attivo l’obiettivo ALL\_ROWS dell’ottimizzatore.

Per conoscere l’ordine delle operazioni, si può utilizzare il comando set autotrace on in modo da vedere il percorso di esecuzione, come descritto nel paragrafo seguente.

## 38.6 Visualizzazione del percorso di esecuzione

Esistono due modi per visualizzare il percorso di esecuzione di una query:

- Usare il comando explain plan.
- Usare il comando set autotrace on.

Nei paragrafi seguenti vengono analizzati entrambi i comandi; nella parte restante del capitolo, il comando set autotrace on verrà utilizzato per illustrare i percorsi di esecuzione così come vengono riportati dall’ottimizzatore.

### Uso di set autotrace on

In SQLPLUS è possibile visualizzare automaticamente il percorso esecutivo per ogni transazione effettuata. Il comando set autotrace on fa in modo che ciascuna query, dopo essere stata eseguita, visualizzi sia il percorso di esecuzione sia informazioni di traccia ad alto livello relative al processo coinvolto nella risoluzione della query.

Per usare il comando set autotrace on è innanzitutto necessario aver creato una tabella PLAN\_TABLE all’interno del proprio account. La struttura di PLAN\_TABLE può cambiare con ogni nuova release di Oracle, quindi si dovrà rimuovere e creare nuovamente la copia di PLAN\_TABLE ogni volta che Oracle viene aggiornato. I comandi mostrati nel listato seguente eliminano qualsiasi PLAN\_TABLE già esistente e la sostituiscono con la versione corrente.

**NOTA** *Per poter utilizzare set autotrace on, il DBA deve aver creato prima il ruolo PLUTRACE nel database e poi aver concesso questo ruolo all’account dell’utente. Il ruolo PLUTRACE consente di accedere alle viste di base, relative alle prestazioni, nel dizionario dati di Oracle. Lo script necessario a creare il ruolo PLUTRACE prende il nome di plustrce.sql e in genere è situato nella directory /sqlplus/admin sotto la directory home del software Oracle.*

L’esempio che segue si riferisce a \$ORACLE\_HOME. Si sostituisca questo simbolo con la directory home del software Oracle sul proprio sistema operativo. Il file che crea la tabella PLAN\_TABLE è situato nella sottodirectory /rdbms/admin della directory home del software Oracle.

```
drop table PLAN_TABLE;
@$ORACLE_HOME/rdbms/admin/utlxplan.sql
```

Quando si utilizza `set autotrace on`, i record vengono inseriti nella copia della tabella `PLAN_TABLE` per mostrare l'ordine delle operazioni eseguite. Terminata la query, vengono visualizzati i dati selezionati. Dopo la visualizzazione dei dati della query, viene mostrato l'ordine delle operazioni, seguito da informazioni statistiche riguardanti l'elaborazione della query. La trattazione seguente relativa al comando `set autotrace on` concentra l'attenzione sulla parte di output che visualizza l'ordine delle operazioni.

**NOTA** *Per visualizzare solo l'output di explain plan, occorre utilizzare il comando `set autotrace on explain`.*

Se si attiva il comando `set autotrace on`, l'explain plan delle query non viene visualizzato fino a che le query non sono state completate. Il comando `explain plan` (descritto successivamente) visualizza i percorsi di esecuzione senza eseguire le query. Quindi, per rendersi conto delle prestazioni di una query, si può attivare il comando `explain plan` prima di eseguirla. Se invece si è quasi sicuri che le prestazioni di una query sono accettabili, si può utilizzare `set autotrace on` per verificarne il percorso di esecuzione.

Nell'esempio seguente, viene effettuata una scansione completa della tabella `BIBLIOTECA`. Le righe dell'output non sono visualizzate in questo output, per brevità. Al di sotto della query viene visualizzato l'ordine delle operazioni.

```
select *
  from BIBLIOTECA;
Execution plan
-----
0      SELECT STATEMENT Optimizer=CHOOSE (Cost=1 Card=31 Byte=1209
)
1      0      TABLE ACCESS (FULL) OF 'BIBLIOTECA' (Cost=1 Card=31 Byte=1
209
```

La sezione "Execution plan" visualizza le fasi intraprese dall'ottimizzatore per eseguire la query. A ciascuna fase viene assegnato un valore di ID (a partire da 0). Il secondo numero visualizza l'operazione "chiamante" dell'operazione corrente. Quindi, nell'esempio precedente la seconda operazione `TABLE ACCESS (FULL) OF BIBLIOTECA` ha un'operazione chiamante (l'istruzione `select`). Ciascuna fase visualizza un costo cumulativo relativo a quella determinata fase, oltre a tutte le fasi secondarie. Si osservi che le interruzioni di linea non vengono mandate a capo; il valore di `Byte` per la fase numero 1 è pari a 1.209.

È anche possibile generare l'ordine delle operazioni dei comandi DML. In questo esempio, viene visualizzato il percorso di esecuzione di un'istruzione `delete`:

```
delete
  from BIBLIOTECA_AUTORE;
Execution plan
-----
0      DELETE STATEMENT Optimizer=CHOOSE (Cost=1 Card=37)
1      0      DELETE OF 'BIBLIOTECA_AUTORE'
2      1      TABLE ACCESS (FULL) OF 'BIBLIOTECA_AUTORE' (Cost=1 Card=3
7)
```

Il comando `delete`, come previsto, richiede la scansione completa di una tabella. Se le tabelle sono state analizzate, l'output della colonna "Execution plan" visualizza il numero di righe di ciascuna tabella, il costo relativo di ciascuna fase e il costo complessivo dell'operazione. Queste informazioni consentono di evidenziare le operazioni più dispendiose per l'elaborazione della query.

Nell'esempio seguente viene eseguita una query leggermente più complessa. Si tratta di una query basata su indice che viene effettuata sulla tabella BIBLIOTECA, che utilizza l'indice di chiave primaria.

```
select /*+ INDEX(biblioteca) */ *
  from BIBLIOTECA
 where Titolo like 'M%';
```

Execution plan

```
-----  
0      SELECT STATEMENT Optimizer=CHOOSE (Cost=2 Card=2 Byte=78)  
1      0      TABLE ACCESS (BY INDEX ROWID) OF 'BIBLIOTECA' (Cost=2 Card=  
2      2      Byte=78)  
2      1      INDEX (RANGE SCAN) OF 'SYS_C002547' (UNIQUE) (Cost=1 Car  
d=2)
```

Questo listato include tre operazioni. Operazione #2, INDEX RANGE SCAN dell'indice di chiave primaria della tabella BIBLIOTECA (il suo nome è stato generato dal sistema), fornisce i dati per l'operazione #1, TABLE ACCESS BY INDEX ROWID. I dati restituiti dall'operazione TABLE ACCESS BY ROWID vengono utilizzati per soddisfare la query (operazione numero 0).

L'output precedente mostra anche che l'ottimizzatore indenta automaticamente ogni fase successiva all'interno del piano esecutivo. In generale, l'elenco delle operazioni andrebbe letto dall'interno verso l'esterno e dall'alto verso il basso. In questo modo, se vengono elencate due operazioni, quella più interna è in genere quella che viene eseguita per prima. Se due operazioni sono sullo stesso livello di indentazione, allora quella elencata per prima (con il numero di operazione più basso) verrà eseguita per prima.

Nell'esempio che segue, vengono unite tramite join le tabelle BIBLIOTECA e BIBLIOTECA\_AUTORE senza l'ausilio di indici:

```
select /*+ USE_MERGE (biblioteca, biblioteca_autore)*/
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA_AUTORE
 where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo
   and BIBLIOTECA.Editore > 'T%';
```

Execution plan

```
-----  
0      SELECT STATEMENT Optimizer=CHOOSE (Cost=7 Card=5 Byte=320)  
1      0      MERGE JOIN (Cost=7 Card=5 Bytes=320)  
2      1      TABLE ACCESS (BY INDEX ROWID) OF 'BIBLIOTECA' (Cost=2 Car  
d=4 Byte=120)  
3      2      INDEX (FULL SCAN) OF 'SYS_C002547' (UNIQUE) (Cost=1 Ca  
rd=4)  
4      1      SORT (JOIN) (Cost=5 Card=37 Byte=1258)  
5      4      TABLE ACCESS (FULL) OF 'BIBLIOTECA_AUTORE' (Cost=1 Card  
=37 Byte=1258)
```

A prima vista, l'indentazione del listato precedente può creare un po' di confusione, ma le informazioni riguardo la gerarchia delle operazioni fornite con la numerazione delle operazioni stesse consentono di chiarire l'ordine di esecuzione. Le operazioni più interne sono quelle che vengono eseguite per prime: TABLE ACCESS FULL e INDEX FULL SCAN. Successivamente, i dati della scansione completa di tabella vengono elaborati tramite un'operazione SORT JOIN (nell'operazione 4), mentre l'output prodotto dalla scansione di indice (che è già ordinato) viene

utilizzato come base per l'operazione TABLE ACCESS BY ROWID (fase 2). La fase 2 e la fase 4 hanno entrambe come operazione chiamante #1, MERGE JOIN. L'operazione MERGE JOIN restituisce i dati all'utente tramite l'istruzione select. L'immagine di questo flusso di dati è già stata mostrata nella Figura 38.5. Non ci dovrebbero essere problemi a leggere il percorso di esecuzione e visualizzare il flusso di dati corrispondenti.

Se la stessa query fosse stata eseguita come un join NESTED LOOPS, sarebbe stato generato un percorso d'esecuzione differente. Come mostrato nel listato seguente, il join NESTED LOOPS è in grado di sfruttare l'indice di chiave primaria che si trova sulla colonna Titolo della tabella BIBLIOTECA:

```
select /*+ INDEX(biblioteca) */
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA_AUTORE
 where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo;
```

Execution plan

```
-----  
0      SELECT STATEMENT Optimizer=CHOOSE (Cost=1 Card=37 Byte=1924  
     )  
1      0      NESTED LOOPS (Cost=1 Card=37 Bytes=1924)  
2      1          TABLE ACCESS (FULL) OF 'BIBLIOTECA_AUTORE' (Cost=1 Card=3  
    7 Byte=1258)  
3      1          INDEX (UNIQUE SCAN) OF 'SYS_C002547' (UNIQUE)
```

I join NESTED LOOPS sono tra i pochi percorsi di esecuzione che non seguono la regola di "lettura dall'interno verso l'esterno", applicata ai percorsi esecutivi indentati. Per leggere il percorso di esecuzione di NESTED LOOPS correttamente, si esamini l'ordine delle operazioni che forniscono direttamente i dati all'operazione NESTED LOOPS (in questo caso la 2 e la 3). Di queste due operazioni, viene eseguita per prima quella con il numero minore (nell'esempio, la 2). Pertanto, l'operazione TABLE ACCESS FULL della tabella BIBLIOTECA\_AUTORE viene eseguita per prima (BIBLIOTECA\_AUTORE è la tabella guida per la query).

Dopo aver stabilito la tabella guida per la query, la parte restante del percorso di esecuzione può essere letta dall'interno verso l'esterno e dall'alto verso il basso. La seconda operazione eseguita è INDEX UNIQUE SCAN dell'indice di chiave primaria della tabella BIBLIOTECA. Quindi, l'operazione NESTED LOOPS è in grado di restituire le righe all'utente.

Se al posto di un join NESTED LOOPS fosse stato selezionato un hash join, il percorso di esecuzione sarebbe stato il seguente:

```
select /*+ USE_HASH (biblioteca) */
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA AUTORE
 where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo;
```

Execution plan

```
-----  
0      SELECT STATEMENT Optimizer=CHOOSE (Cost=3 Card=37 Byte=1924  
     )  
1      0      HASH JOIN (Cost=3 Card=37 Byte=1924)  
2      1          INDEX (FULL SCAN) OF 'SYS_C002547' (UNIQUE) (Cost=1 Card  
    =31 Byte=558)  
3      1          TABLE ACCESS (FULL) OF 'BIBLIOTECA_AUTORE' (Cost=1 Card=3  
    7 Byte=1258)
```

Il percorso esecutivo dell'hash join mostra che sono state eseguite due scansioni separate. Dal momento che le due operazioni compaiono allo stesso livello di indentazione, la scansione dell'indice di chiave primaria della tabella BIBLIOTECA viene eseguita per prima (in quanto possiede un numero d'operazione più basso).

**NOTA** *Quando si utilizza il comando set autotrace on, non è necessario gestire i record all'interno della tabella PLAN\_TABLE. Oracle cancella automaticamente i record che inserisce in PLAN\_TABLE dopo aver visualizzato il percorso di esecuzione.*

Se si utilizzano le opzioni per effettuare le query in parallelo o si eseguono query su database remoti, una parte aggiuntiva dell'output di set autotrace on mostra il testo delle query eseguite dai processi del server di query parallele, oppure la query eseguita all'interno del database remoto. Per disattivare la funzionalità di autotrace, è sufficiente eseguire il comando set autotrace off.

## Uso di explain plan

Il comando explain plan può essere utilizzato per generare il percorso di esecuzione per una query senza eseguirla. Per utilizzare questo comando è necessario aver creato in precedenza una tabella PLAN\_TABLE all'interno del proprio schema (fare riferimento alle istruzioni precedenti per la creazione della tabella). Per determinare il percorso di esecuzione di una query è necessario anteporre alla query stessa il seguente codice SQL:

```
explain plan
set Statement_ID = 'TEST'
for
```

Per semplificare il processo di messa a punto, si utilizzi sempre lo stesso valore di Statement\_ID e si cancellino i record per ciascun percorso di esecuzione prima di utilizzare una seconda volta il comando explain plan.

Nel listato seguente, viene riportato un esempio di esecuzione del comando explain plan. La query mostrata nel listato non viene eseguita durante il comando; vengono generate solamente le fasi del suo percorso esecutivo, successivamente inserite come record in PLAN\_TABLE.

```
explain plan
set Statement_ID = 'TEST'
for
select /*+ USE_HASH (biblioteca) */
       BIBLIOTECA_AUTORE.NomeAutore
  from BIBLIOTECA, BIBLIOTECA_AUTORE
 where BIBLIOTECA.Titolo = BIBLIOTECA_AUTORE.Titolo;
```

Explained.

Quando si esegue il comando explain plan, i record vengono inseriti nella tabella PLAN\_TABLE. È possibile eseguire una query sulla tabella PLAN\_TABLE con la query seguente. I risultati della query mostrano le operazioni eseguite ad ogni fase e le relazioni padre-figlio tra le fasi del percorso esecutivo, in modo analogo alla visualizzazione indentata del comando set autotrace on:

```
select ID ID_plus_exp,
Padre_ID parent_id_plus_exp,
```

```

LPAD(' ',2*(level-1))|| /* Indenta per il livello */
Operation|| /* L'operazione */
DECODE(other_tag,null,'','*')|| /* visualizza un segno '*' se parallela */
DECODE(options,null,'',' ('||options||')')|| /* visualizza le opzioni */
DECODE(object_name,null,'',' of ''||object_name||'')|| 
DECODE(object_type,null,'',' ||object_type||')|| 
DECODE(id,0,decode(optimizer,null,' optimizer'||optimizer))|| 
DECODE(cost,null,'.' (cost='||cost|| /* visualizza informazioni sui costi. */ 
DECODE(cardinality,null,'.' card='||cardinality)|| /* cardinalità */
DECODE(byte,null,'.' byte='||byte)||')) plan_plus_exp.
object_node object_node_plus_exp /* informazioni parallele e remote */
from PLAN_TABLE
start with ID=0 and Statement_ID='TEST'
connect by prior ID=Parent_ID and Statement_ID='TEST'
order by ID,Position;

```

La query visualizzata nel listato precedente utilizza l'operatore connect by per valutare la gerarchia delle fasi nel percorso esecutivo della query. La query del listato presuppone che il campo Statement\_ID sia impostato al valore 'TEST'. Le fasi del percorso esecutivo vengono visualizzate nella colonna cui è stato assegnato l'alias "Execution\_Path".

Nel listato che segue viene mostrato l'output della query precedente:

i	p PLAN_PLUS_EXP	OBJECT_N
0	SELECT STATEMENT optimizer=CHOOSE (cost=3 card=37 byte=1924 )	
1	0 HASH JOIN (cost=3 card=37 byte=1924)	
2	1 INDEX (FULL SCAN) of 'SYS_C002547' UNIQUE (cost=1 card= 31 byte=558)	
3	1 TABLE ACCESS (FULL) of 'BIBLIOTECA_AUTORE' (cost=1 card=3 7 byte=1258)	

Le prime due colonne mostrano il valore di ID e l'ID padre per ogni fase. L'output include la colonna Cost, che visualizza il costo relativo di ogni fase. Durante il processo di messa a punto non è possibile confrontare i costi di query differenti.

### **Implementazione di strutture memorizzate**

Mentre si esegue la migrazione da un database all'altro, i percorsi esecutivi delle query possono cambiare. I percorsi di esecuzione possono cambiare per diversi motivi:

- L'uso di un ottimizzatore diverso in database differenti (basato sui costi in un database, basato sulle regole nell'altro).
- L'abilitazione di caratteristiche dell'ottimizzatore diverse in database differenti.
- Le statistiche per le tabelle su cui è stata eseguita la query possono differire nei database.
- La frequenza con cui le statistiche vengono raccolte può differire tra i vari database.
- I database possono eseguire versioni differenti del kernel Oracle.

Gli effetti di queste differenze sui percorsi esecutivi possono essere notevoli, e possono influire in modo negativo sulle prestazioni delle query mentre si esegue la migrazione oppure mentre si aggiorna l'applicazione. Per ridurre al minimo l'impatto di queste differenze sulle prestazioni delle query, Oracle ha introdotto con Oracle8i una caratteristica di nome *struttura memorizzata*.

Una struttura memorizzata memorizza un gruppo di suggerimenti per una query. Questi suggerimenti verranno utilizzati ogni volta che si esegue la query. L'uso di suggerimenti memorizzati aumenta la probabilità che la query utilizzi sempre lo stesso percorso esecutivo. I suggerimenti non impongono un determinato percorso esecutivo (sono suggerimenti e non comandi), tuttavia riducono l'impatto degli spostamenti del database sulle prestazioni delle query.

Per iniziare a creare dei suggerimenti per tutte le query, si imposti il parametro di inizializzazione `CREATE_STORED_OUTLINES` a TRUE; a questo punto tutte le strutture verranno salvate nella categoria `DEFAULT`. In alternativa, esiste la possibilità di creare categorie personalizzate di strutture e utilizzare il nome della categoria come valore nel file dei parametri di inizializzazione, come mostrato nel listato seguente:

```
CREATE_STORED_OUTLINES = sviluppo
```

In questo esempio, le strutture memorizzate verranno memorizzate per query contenute nella categoria `SVILUPPO`.

Per creare una struttura, è necessario disporre del privilegio di sistema `CREATE ANY OUTLINE`. Si utilizzi il comando `create outline` per creare una struttura per una query, come mostrato nel listato seguente:

```
create outline RIORDINI
  for category SVILUPPO
  on
select BIBLIOTECA.Titolo
  from BIBLIOTECA, LIBRO_ORDINE
 where BIBLIOTECA.Titolo = LIBRO_ORDINE.Titolo;
```

**NOTA** *Se non si specifica un nome per la struttura, le verrà assegnato un nome generato dal sistema.*

Se `CREATE_STORED_OUTLINES` è stato impostato a TRUE nel file dei parametri di inizializzazione, l'RDBMS creerà delle strutture memorizzate per le query; l'uso del comando `create outline` offre un maggior controllo sulle strutture create.

**NOTA** *È possibile creare delle strutture per i comandi DML e per i comandi create table as select.*

Dopo aver creato una struttura, la si potrà modificare. Per esempio, potrebbe essere necessario alterare la struttura in modo che rifletta le modifiche significative nei volumi e nella distribuzione dei dati. Per generare di nuovo i suggerimenti adoperati durante l'esecuzione della query si può ricorrere alla clausola `rebuild` del comando `alter outline`:

```
alter outline RIORDINI rebuild;
```

Sempre con la clausola `rename` del comando `alter outline` è possibile rinominare una struttura:

```
alter outline RIORDINI rename to BIBLIOTECA_RIORDINI;
```

Con la clausola `change category` è possibile modificare la categoria di una struttura, come mostrato nell'esempio seguente:

```
alter outline RIORDINI change category to DEFAULT;
```

Per permettere all'ottimizzatore di usare le strutture memorizzate, si imposti il parametro di inizializzazione USE\_STORED\_OUTLINES a TRUE oppure al nome di una categoria (come SVILUPPO negli esempi precedenti). Se l'uso delle strutture memorizzate è stato abilitato, qualsiasi query con una struttura memorizzata utilizzerà i suggerimenti creati insieme alla struttura. Inoltre, è possibile abilitare USE\_STORED\_OUTLINES a livello della sessione con il comando alter session.

Per gestire le strutture memorizzate, si utilizza il package OUTLN\_PKG, che offre tre possibilità:

- Eliminare le strutture che non sono mai state utilizzate.
- Eliminare le strutture con una categoria specifica.
- Spostare le strutture da una categoria all'altra.

Ciascuna di queste tre possibilità ha una procedura corrispondente in OUTLN\_PKG. Per eliminare le strutture che non sono mai state utilizzate, si esegua la procedura DROP\_UNUSED, come mostrato nel listato seguente:

```
execute OUTLN_PKG.DROP_UNUSED;
```

Per eliminare tutte le strutture contenute in una determinata categoria, si esegua la procedura DROP\_BY\_CAT, che come unico parametro di input ha il nome della categoria. L'esempio seguente elimina tutte le strutture contenute nella categoria SVILUPPO:

```
execute OUTLN_PKG.DROP_BY_CAT -
(cat => 'DEVELOPMENT');
```

Per assegnare nuovamente le strutture da una vecchia categoria a una nuova, si esegua la procedura UPDATE\_BY\_CAT, come mostrato nell'esempio seguente:

```
execute OUTLN_PKG.UPDATE_BY_CAT -
(old_cat => 'SVILUPPO', -
new_cat => 'TEST');
```

Per eliminare una struttura specifica, si utilizzi il comando drop outline.

## 38.7 Operazioni varie

Oltre alle operazioni presentate in precedenza nel capitolo, accesso alle tabelle, accesso agli indici, manipolazioni di gruppi di dati e unioni, esistono altre operazioni che vengono utilizzate dall'ottimizzatore nell'esecuzione di una query. Le operazioni descritte nei paragrafi seguenti possono influire sulle prestazioni delle query, ma tendono ad essere messe a punto e ad essere utilizzate in misura minore rispetto a quelle descritte precedentemente.

### Filtro di righe

Quando si usa una clausola `where` per eliminare righe da una query, Oracle può utilizzare un'operazione FILTER per selezionare le righe che corrispondono ai criteri della clausola `where`. L'operazione FILTER non viene visualizzata sempre nell'output prodotto dal comando set autotrace on, anche se può essere stata eseguita. Per un'illustrazione dell'operazione FILTER, si rimanda all'esempio della vista proposto in precedenza in questo capitolo. Quando un'operazione FILTER

viene elencata esplicitamente nel percorso di esecuzione di una query, solitamente sta a indicare che non è disponibile alcun indice per assistere l'elaborazione di una condizione di limitazione. Le operazioni FILTER si vedono comunemente in query che utilizzano la clausola connect by e occasionalmente durante l'elaborazione delle operazioni VIEW.

## Query che utilizzano clausole connect by

Quando in una query viene impiegata una clausola connect by, Oracle ricorre a un'operazione CONNECT BY per eseguire la query. Il percorso di esecuzione mostra che l'operazione CONNECT BY accetta solitamente tre input mentre avanza verso l'alto e verso il basso all'interno di un "albero" di record. Per ottenere le prestazioni migliori da una query connect by è opportuno creare un gruppo di indici che possano essere utilizzati dalle colonne nella clausola connect by.

Gli esempi di clausole connect by mostrati in questo libro utilizzano la tabella ALLEVAMENTO per illustrare l'albero genealogico di un allevamento di bovini. La query seguente percorre l'albero genealogico a partire dal capostipite (la mucca di nome 'EVE') per giungere ai discendenti. Può essere specificata una clausola connect by per spostarsi dai progenitori ai discendenti, o dai discendenti agli avi.

```
select Mucca,
       Toro,
       LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sesso,
       Datanascita
  from ALLEVAMENTO
 start with Figlio = 'EVE'
connect by Mucca = PRIOR Figlio;
```

Senza indici sulla tabella ALLEVAMENTO, per questa query viene generato il seguente percorso esecutivo (come riportato dal comando set autotrace on, con la tabella analizzata):

### Execution Plan

```
-----
0      SELECT STATEMENT Optimizer=CHOOSE (Cost=1 Card=16 Byte=320)
1      0      CONNECT BY (WITH FILTERING)
2      1      NESTED LOOPS
3      2          TABLE ACCESS (FULL) OF 'ALLEVAMENTO' (Cost=1 Card=1 Byte
   =5)
4      2          TABLE ACCESS (BY USER ROWID) OF 'ALLEVAMENTO'
5      1      HASH JOIN
6      5      CONNECT BY PUMP
7      5          TABLE ACCESS (FULL) OF 'ALLEVAMENTO' (Cost=1 Card=16 Byte
   s=320)
```

L'operazione CONNECT BY presenta più input: una scansione iniziale di tabella per stabilire la posizione del nodo principale, seguita da un paio di operazioni di accesso alla tabella per attraversare l'albero dei dati. Delle operazioni di accesso ai dati che forniscono dati all'operazione CONNECT BY, due sono operazioni TABLE ACCESS FULL. Per ridurre il numero di operazioni TABLE ACCESS FULL richieste dall'operazione CONNECT BY, si dovrebbe creare una coppia di indici concatenati sulle colonne utilizzate nella clausola connect by. Per la query della tabella ALLEVAMENTO, la clausola connect by è:

```
connect by Mucca = PRIOR Figlio
```

Per indicizzare in modo appropriato i dati per la query, si possono creare due indici concatenati sulle colonne Mucca e Figlio. Nel primo indice, la colonna Mucca dovrebbe essere quella principale. Nel secondo indice, va invertito l'ordine delle colonne. La coppia di indici creata sulle colonne Mucca e Figlio può essere utilizzata indipendentemente dalla direzione intrapresa per scorrere l'albero dei dati. A partire da Oracle9i, si potrà creare soltanto un indice concatenato e usare la caratteristica skip-scan per eseguire una scansione rapida dell'indice.

Nel listato che segue, vengono creati due indici concatenati:

```
create index Mucca$Figlio
on ALLEVAMENTO(Mucca, Figlio);
create index Figlio$Mucca
on ALLEVAMENTO(Figlio, Mucca);
```

Avendo a disposizione questi due indici, il percorso esecutivo per la query della tabella ALLEVAMENTO utilizza accessi basati sugli indici, anziché scansioni complete della tabella, come mostrato nel listato seguente. Si osservi che potrebbe essere necessario suggerire all'ottimizzatore di utilizzare l'indice specifico, come mostrato di seguito:

```
select /*+ INDEX(allevamento figlio$mucca) */
       Mucca,
       Toro,
       LPAD(' ',6*(Level-1))||Figlio  Figlio,
       Sesso,
       DataNascita
  from ALLEVAMENTO
 start with Figlio = 'EVE'
connect by Mucca = PRIOR Figlio;
```

#### Execution Plan

```
-----
0      SELECT STATEMENT Optimizer=CHOOSE (Cost=148 Card=3 Byte=60)
1      0      CONNECT BY (WITH FILTERING)
2      1      NESTED LOOPS
3      2          TABLE ACCESS (FULL) OF 'ALLEVAMENTO' (Cost=1 Card=1 Byte
=5)
4      2          TABLE ACCESS (BY USER ROWID) OF 'ALLEVAMENTO'
5      1      NESTED LOOPS
6      5          BUFFER (SORT)
7      6          CONNECT BY PUMP
8      5          TABLE ACCESS (BY INDEX ROWID) OF 'ALLEVAMENTO' (Cost=148
Card=3 Byte=60)
9      8              INDEX (SKIP SCAN) OF 'FIGLIO$MUCCA' (NON-UNIQUE) (C
ost=18 Card=3)
```

In questo listato, una delle operazioni TABLE ACCESS FULL per la tabella ALLEVAMENTO è stata sostituita da accessi basati su indice. È stato utilizzato un suggerimento per indurre l'ottimizzatore a utilizzare l'indice FIGLIO\$MUCCA, mostrando così la caratteristica skip scan di Oracle9i (fare riferimento all'operazione #9 nel listato). In generale, gli accessi basati su indice hanno prestazioni migliori rispetto alle scansioni complete di tabelle per le query connect by, soprattutto se esistono più livelli nell'albero dei dati.

## Query che utilizzano sequenze

Quando una query seleziona valori da una sequenza, compare l'operazione SEQUENCE nel percorso esecutivo della query. In generale, i valori delle sequenze vengono selezionati effettu-

tuando una query su DUAL (il sinonimo pubblico della tabella SYSDUAL). Se un'altra tabella è già coinvolta nella query, non è necessario aggiungere DUAL alla query per eseguire la selezione dalla sequenza.

L'impiego di DUAL come tabella base per la query è arbitrario; i criteri importanti sono che la tabella sia accessibile e che possieda una riga per ogni numero della sequenza che si desidera restituire.

Per migliorare le prestazioni delle query che generano valori dalle sequenze, è possibile inserire nella cache un gruppo preallocato di valori di sequenza. Per default, possono essere memorizzati nella cache 20 valori di sequenza. Durante questa operazione, Oracle non riempie la cache fino a che tutti e 20 i valori sono stati utilizzati. Se il database viene arrestato, qualsiasi valore della sequenza contenuto nella cache viene perso; durante il successivo avvio del database, verranno registrati nella cache i 20 valori successivi.

Per esempio, se si selezionano 12 valori da una sequenza, 8 rimarranno nella cache. Se poi si chiude il database, questi otto valori vengono persi. Quando si riavvia il database, viene memorizzata nella cache un'altra sequenza di 20 valori, a partire dall'ultimo valore precedentemente inserito nella cache. Quindi ci sarà un'interruzione di otto valori della sequenza nei valori selezionati dalla sequenza. Per la sintassi dei comandi `create sequence` e `alter sequences`, si rimanda al Capitolo 42.

## **Query che utilizzano database link**

Se una query utilizza un database link per accedere ai dati remoti, l'ottimizzatore mostra che è stata eseguita un'operazione REMOTE. Quando si effettua un'operazione REMOTE, nella colonna Other della tabella PLAN\_TABLE viene registrata la query eseguita nel database remoto. Nella messa a punto di una query che utilizza oggetti remoti, è necessario effettuare operazioni di SEEK per ridurre la quantità di traffico tra le istanze. La riduzione del traffico tra i database comprende sia la limitazione della dimensione dei dati inviati dal database remoto a quello locale, sia la riduzione del numero di accessi al database remoto durante la query.

## **Query che utilizzano cluster**

Se una tabella è memorizzata in un cluster, per accedervi si può utilizzare l'operazione TABLE ACCESS CLUSTER. In generale, la manipolazione di dati su tabelle associate a cluster ha prestazioni inferiori rispetto alle manipolazioni di dati su tabelle non associate a cluster. Se le tabelle vengono modificate spesso tramite `insert`, `update` e `delete`, l'utilizzo dei cluster può influire negativamente sulle prestazioni dell'applicazione. I cluster sono maggiormente indicati quando i dati memorizzati al loro interno sono statici.

Per l'esempio riguardante l'unione NESTED LOOPS, la tabella BIBLIOTECA\_AUTORE è stata utilizzata come tabella guida ed è stato eseguito un accesso basato su indice per trovare i record corrispondenti nella tabella BIBLIOTECA. Durante l'accesso ai dati di BIBLIOTECA è stato utilizzato l'indice di chiave primaria. Se la tabella BIBLIOTECA fosse stata memorizzata in un cluster, con la colonna Titolo come colonna chiave del cluster, le operazioni eseguite per accedere ai dati all'interno del join NESTED LOOPS sarebbero state leggermente diverse.

Se BIBLIOTECA fosse in un cluster, la tabella BIBLIOTECA\_AUTORE potrebbe essere ancora la tabella guida per il join. L'accesso ai dati della tabella BIBLIOTECA potrebbe avvenire in due fasi: un'operazione INDEX UNIQUE SCAN dell'indice di chiave del cluster, seguita da un'operazione TABLE ACCESS CLUSTER della tabella BIBLIOTECA. L'operazione TABLE ACCESS CLUSTER viene utilizzata ogni volta che i dati vengono letti da una tabella dopo aver selezionato i valori dall'indice del cluster della tabella stessa.

Se BIBLIOTECA si trovasse in un cluster hash, il metodo di lettura dei dati dalla tabella cambierebbe completamente. In un cluster hash la posizione fisica di una riga è determinata dai valori della riga stessa. Se BIBLIOTECA si trovasse in un cluster hash e i valori della colonna Titolo venissero utilizzati come funzione di hash per il cluster hash, la query:

```
select *
  from BIBLIOTECA
 where Titolo = 'INNUMERACY';
```

potrebbe utilizzare un'operazione TABLE ACCESS HASH per leggere i dati. Dal momento che la collocazione fisica dei dati dovrebbe essere già nota all'ottimizzatore, non è necessario alcun accesso basato su indice.

## Suggerimenti

È possibile utilizzare il suggerimento HASH per comunicare all'ottimizzatore di ricorrere a un'operazione TABLE ACCESS HASH mentre esegue la scansione di una tabella. Quando si specifica il suggerimento HASH, si dovrebbe indicare il nome (o l'alias) della tabella che dovrà essere letta tramite l'operazione TABLE ACCESS HASH.

**NOTA** *Non esiste alcuna relazione tra i cluster hash e gli hash join. Per specificare l'uso di hash join, si utilizza il suggerimento USE\_HASH.*

Se si usano cluster non hash, con il suggerimento CLUSTER si può specificare che una tabella verrà letta tramite un'operazione TABLE ACCESS CLUSTER. In generale, i suggerimenti CLUSTER e HASH vengono utilizzati principalmente quando le query in questione non contengono join. Se una query comprende un join, si dovrebbero utilizzare i suggerimenti relativi ai join, in modo da avere l'impatto maggiore sul percorso esecutivo scelto per la query.

## Altri aspetti di messa a punto

Oltre alle operazioni e ai relativi suggerimenti citati nei paragrafi precedenti, esistono altri due gruppi di suggerimenti che possono influire sull'elaborazione di una query. Questi suggerimenti aggiuntivi consentono di controllare l'esecuzione parallela delle query e le memorizzazioni dei dati nella cache all'interno della SGA.

Se il server dispone di più processori e i dati su cui si dovrà eseguire la query sono distribuiti su più periferiche, è possibile migliorare l'elaborazione delle query implementando l'accesso ai dati e le operazioni di ordinamento in parallelo. Quando una query viene eseguita in parallelo, vengono attivati in parallelo più processi, ciascuno dei quali accede ai dati o li ordina. La distribuzione del carico di lavoro e l'assemblaggio dei risultati di ciascuna query è a carico di un processo di coordinamento.

Il *grado di parallelismo*, ovvero il numero di processi di scansione o ordinamento avviati per una query, può essere impostato a livello di tabella (fare riferimento al comando `create table` nel Capitolo 42). L'ottimizzatore rileva le impostazioni a livello di tabella relative al grado di parallelismo e stabilisce il numero di processi sul server da utilizzare per risolvere la query. L'ottimizzatore è in grado di controllare dinamicamente il numero di processori e dispositivi coinvolti nella query e di basare le proprie decisioni riguardo al parallelismo sulle risorse disponibili.

È possibile influire sul parallelismo delle query tramite il suggerimento PARALLEL. Nell'esempio seguente, è stato impostato un grado di parallelismo pari a 4 per una query sulla tabella BIBLIOTECA:

---

```
select /*+ PARALLEL (biblioteca,4) */ *
  from BIBLIOTECA;
```

Se si utilizza l'opzione Real Application Clusters, è possibile specificare un parametro "istante" nel suggerimento PARALLEL. Nella query seguente, viene eseguito in parallelo il processo di scansione completa della tabella, con un grado di parallelismo suggerito pari a 4 e distribuito su due istanze:

```
select /*+ PARALLEL(biblioteca,4,2) */ *
  from BIBLIOTECA
 order by Editore;
```

Nella query precedente, è stata aggiunta una seconda operazione, SORT ORDER BY, per la clausola order by Editore. Dal momento che è possibile eseguire in parallelo anche l'operazione di ordinamento, la query può utilizzare nove processi sul server anziché cinque (quattro per la scansione della tabella, quattro per l'ordinamento e uno per il coordinamento della query). L'ottimizzatore stabilisce in modo dinamico il grado di parallelismo da utilizzare basandosi sulle impostazioni della tabella, sul layout del database e sulle risorse di sistema disponibili.

Se si desidera disattivare l'esecuzione in parallelo di una query, è possibile usare il suggerimento NOPARALLEL. Questo suggerimento può essere utilizzato per eseguire in modo seriale una query su una tabella per la quale le query vengono eseguite normalmente in parallelo.

Se una tabella piccola (con una dimensione inferiore a cinque blocchi di database) viene letta tramite una scansione completa, i suoi dati vengono contrassegnati per essere mantenuti il più a lungo possibile nella SGA. Anche i dati letti tramite scansioni di indice e operazioni TABLE ACCESS BY ROWID vengono mantenuti nella SGA il più a lungo possibile e vengono rimossi solo quando viene richiesta della memoria aggiuntiva dall'applicazione. Se una tabella ha una dimensione maggiore di cinque blocchi e viene letta con una scansione completa, in genere i suoi blocchi vengono contrassegnati per essere rimossi dalla SGA al più presto possibile. Se la tabella letta con una scansione completa ha una dimensione maggiore di cinque blocchi, ma si desidera mantenerne i dati in memoria il più a lungo possibile, è possibile contrassegnare la tabella come "tabella inserita nella cache" usando la clausola cache del comando create table oppure del comando alter table.

Se una tabella di grandi dimensioni non è stata contrassegnata come tabella "inserita nella cache", e si desidera fare in modo che i suoi dati restino nella SGA dopo che la query è stata completata, si può utilizzare il suggerimento CACHE per comunicare all'ottimizzatore di mantenere i dati nella SGA il più a lungo possibile. Questo suggerimento in genere viene utilizzato insieme a FULL.

Nel listato seguente viene eseguita una query sulla tabella BIBLIOTECA; anche se i dati vengono letti tramite un'operazione TABLE ACCESS FULL, non vengono rimossi immediatamente dalla SGA.

```
select /*+ FULL(biblioteca) CACHE(biblioteca) */ *
  from BIBLIOTECA;
```

Se una tabella viene utilizzata spesso da molte query o utenti, e non esiste uno schema di indicizzazione appropriato disponibile, è consigliabile utilizzare la cache per migliorare le prestazioni relative agli accessi alla tabella. L'utilizzo della cache risulta particolarmente utile per tabelle statiche. Per disattivare in modo dinamico le impostazioni della cache per una tabella, si può utilizzare il suggerimento NOCACHE.

## 38.8 Conclusioni

Nei paragrafi precedenti sono stati illustrati i metodi utilizzati dall'ottimizzatore per accedere ai dati e per operare su di essi. Nella messa a punto delle query, vanno tenuti presenti diversi fattori.

- **L'obiettivo dell'utente** Si vuole fare in modo che le righe vengano restituite velocemente, oppure l'attenzione maggiore è rivolta al tempo complessivo di risoluzione della query?
- **Il tipo di join utilizzati** Le tabelle sono indicizzate in modo appropriato per avvantaggiarsi delle operazioni NESTED LOOPS?
- **La dimensione delle tabelle** Quanto sono grandi le tabelle per cui si esegue la query? Se vi sono più join, quant'è grande la mole di dati restituita da ciascuno?
- **La selettività dei dati** Quanto sono selettivi gli indici utilizzati?
- **Le operazioni di ordinamento** Quante operazioni di ordinamento vengono eseguite? Vi sono operazioni di ordinamento annidate?

Se si riescono a gestire in modo accurato gli aspetti relativi alle prestazioni, il numero di query "problematiche" all'interno del proprio database dovrebbe diminuire. Se ci si accorge che una query richiede più risorse di quante dovrebbe, è consigliabile utilizzare i comandi set autotrace on o explain plan per stabilire l'ordine delle operazioni utilizzate per la query. Si tengano presenti i suggerimenti e le informazioni forniti nel corso di questo capitolo per la successiva messa a punto delle operazioni o per la loro impostazione. Una query messa a punto in questo modo sarà in grado di soddisfare gli obiettivi prefissati.



## Capitolo 39

# Guida a Oracle9iAS

- 39.1 **Communication Services**
- 39.2 **Content Management Services**
- 39.3 **Ultra Search**
- 39.4 **Business Logic Services**
- 39.5 **Presentation Services**
- 39.6 **Business Intelligence Services**
- 39.7 **Portal Services**
- 39.8 **Kit per sviluppatori**
- 39.9 **Caching Services**
- 39.10 **System Services**
- 39.11 **Database Services**
- 39.12 **Avvio, arresto e riavvio di iAS (Apache)**

**C**on l'introduzione di Oracle9i Application Server (9iAS), Oracle presenta alla comunità degli sviluppatori una suite completa di strumenti e applicazioni per lo sviluppo, la distribuzione, la gestione e la messa a punto di applicazioni per il Web. Oracle9iAS è un application server di livello intermedio affidabile e scalabile, che consente a utenti e organizzazioni di sfruttare a pieno le potenzialità di Internet. Invece di configurarsi come un singolo prodotto, Oracle9iAS è una collezione di servizi concepiti per collaborare allo scopo di offrire contenuto sul Web in modo dinamico. Si possono selezionare e scegliere i servizi necessari, e aggiungerne di nuovi che assecondano la propria crescita e i vari cambiamenti. Oracle9iAS è concepito per consentire l'impiego dei metodi, degli strumenti e degli standard più recenti per lo sviluppo Web grazie al supporto di Java J2EE, XML, PL/SQL o Java Server Pages, nonché di molte altre tecnologie. Oracle9iAS fornisce anche servizi per la configurazione di altri servizi per applicazioni, servizi per la configurazione del caching del Web e dei database, e moltissimi altri componenti per applicazioni.

**NOTA** *Si ringrazia Brad Brown della società TUSC per il suo contributo a questo capitolo. Per ulteriori dettagli sullo sviluppo delle applicazioni basate su 9iAS, è possibile fare riferimento al manuale di Brad dal titolo Oracle9i Sviluppo Web (McGraw-Hill, 2002).*

Oracle classifica i servizi di 9iAS in diversi gruppi in base al tipo di supporto offerto:

- **Communication Services** Web server per gestire le richieste a e dal Web.
- **Business Logic Services** Strumenti e linguaggi di sviluppo per la creazione di applicazioni personalizzate.
- **Presentation Services** Strumenti per lo sviluppo di pagine Web dinamiche.
- **Caching Services** Strumenti per il miglioramento delle prestazioni dei siti Web grazie al caching dei dati o delle pagine comuni.

- **Content Management Services** Internet File System (IFS) per la gestione di documenti nel database.
- **Business Intelligence Services** Utili per la creazione di report e query ad hoc delle attività che gli utenti svolgono nei siti.
- **Database Services** Database Oracle9*i* per la memorizzazione dei dati delle applicazioni.

Nella suite 9iAS, Oracle fornisce anche diversi kit per sviluppatori che aiutano a creare applicazioni dinamiche e flessibili, e strumenti per la gestione del database e della sicurezza una volta che i siti Web sono stati pubblicati.

La grande apertura di questa suite offre ai progettisti e agli sviluppatori di siti una grande flessibilità e scalabilità negli strumenti e nei servizi utilizzabili. Quando le esigenze crescono, è possibile aggiungere servizi o cambiare la direzione di sviluppo senza lasciare l'ambiente integrato di Oracle. Per esempio, se si è pratici di Oracle Forms, ma non si ha molta esperienza con il Web, si potrebbe scegliere inizialmente di procedere soltanto alla distribuzione dei moduli e dei report sul Web, per poi aggiungere codice Java o PL/SQL Server Pages una volta acquisita una maggiore esperienza. Nel caso in cui si desideri rendere dinamiche alcune pagine statiche, è possibile guadagnare un certo controllo sulla gestione del contenuto statico utilizzando IFS, procedere alla distribuzione e aggiungere in un secondo tempo delle tag Java per creare pagine JSP (Java Server Pages) che interrogheranno il contenuto dinamico a partire dal database. Quando il traffico sul sito aumenta, potrebbe essere vantaggioso procedere al caching dei dati o delle pagine più usate per aumentare le prestazioni. Quando i siti Web diventano *davvero* molto noti, si può procedere alla distribuzione delle applicazioni e dei dati su più server, aggiungere altro caching e gestire il tutto con i servizi di sistema Enterprise Manager e Advanced Security. Ovviamente, quando i siti guadagnano prestigio e sono oggetto di molta attenzione, ci saranno altri che desidereranno incorporarne il contenuto, cosa che si può facilmente ottenere ricorrendo a XML e ad Enterprise Java Beans.

Attualmente esistono tre edizioni di Oracle9iAS: Standard Edition, Enterprise Edition e Wireless Edition. La *Standard Edition* fornisce gli strumenti necessari per iniziare i lavori di sviluppo delle applicazioni e per crescere fino a diventare un sito Web di medie dimensioni. La *Enterprise Edition* offre servizi aggiuntivi che consentono a un sito Web di aumentare notevolmente la propria presenza. Per la distribuzione di contenuto destinato ai telefoni cellulari e alle piccole applicazioni Web che usano l'interfaccia Wireless Access Protocol (WAP), Oracle fornisce la *Wireless Edition*, che aggiunge il componente Portal-to-Go.

In 9iAS release 2, Oracle ha migliorato ulteriormente la funzionalità del toolset 9iAS e ha diviso in nuove categorie alcune soluzioni. In 9iAS release 2, le soluzioni e i componenti sono i seguenti:

SOLUZIONE	COMPONENTI ORACLE9IAS
J2EE e applicazioni Internet	Oracle HTTP Server Oracle9iAS Containers for J2EE Oracle9iAS Web Services Oracle PL/SQL Oracle9iAS Forms Services Oracle XML Developer Kit Oracle9i Client
Portali	Oracle9iAS Portal Oracle9iAS Portal Developer Kit
Wireless	Oracle9iAS Wireless

(segue)

*(continua)*


---

Caching	Oracle9iAS Web Cache
Business Intelligence	Oracle9iAS Reports Services Oracle9iAS Discoverer Oracle9iAS Personalization Oracle9iAS Clickstream Intelligence
E-Business Integration	Oracle9iAS InterConnect Oracle9iAS Unified Messaging Oracle Internet File System
Gestione e sicurezza	Oracle Enterprise Manager Oracle9iAS Single Sign-On Oracle Internet Directory Oracle9iAS Infrastructure

---

**NOTA** *A partire dalla Release 2, Oracle9iAS Database Cache non fa più parte di 9iAS.*

Nei paragrafi seguenti verranno descritti i componenti principali di 9iAS. Non si tratta di un elenco completo, bensì di una descrizione qualitativamente valida delle caratteristiche più utilizzate. Per descrizioni dettagliate di tutti i componenti si faccia riferimento alla documentazione di produzione completa.

## 39.1 Communication Services

I *Communication Services* forniscono la connettività a Internet agli utenti. Oggi l'aspetto e l'interfaccia di Internet più visibile è il Web, costituito da computer client e server in grado di gestire documenti multimediali. Internet segue un insieme di protocolli e convenzioni standard ed è divenuta un universo di informazioni accessibili tramite rete. La comunicazione sul Web si basa sull'Hypertext Transfer Protocol (HTTP). I Communication Services sono costituiti dai componenti, protocolli e procedure che forniscono servizi di comunicazione multimediale, come audio in tempo reale, video e comunicazione di dati, su reti a pacchetti, che includono le reti basate sull'Internet Protocol (IP). L'aspetto più importante è che i Communication Services sono, in pratica, il Web server.

Oracle9iAS fornisce diverse opzioni per i Communication Services. Questi servizi continuano ad ampliarsi con ogni release di 9iAS.

### Communication Services – Powered by Apache

Nella release corrente, Oracle supporta i seguenti moduli Apache di Oracle:

mod_ssl	Supporto per la cifratura dei messaggi con Secure Sockets Layer (SSL).
mod_perl	Supporto per la scrittura di moduli Apache in Perl.
mod_jserv	Comunicazione con un servlet engine Java (J2EE).
mod_plsql	Supporto per il toolkit PL/SQL.

---

mod_ose	Delega gli URL a servlet Java con conservazione dello stato e PL/SQL nell'Oracle Servlet Engine (OSE). Questo modulo mantiene gli ID di sessione in cookie o in URL reindirizzati, instrada le richieste alle sessioni OSE appropriate e comunica con l'OSE tramite Network Services (ossia, SQL*Net), che fornisce supporto firewall tra il server HTTP di Oracle e OSE e connection pooling.
http_core	Caratteristiche fondamentali di Apache.
mod_access	Controllo degli accessi basato su host; offre un controllo degli accessi basato sul nome di host o sull'indirizzo IP del client.
mod_actions	Esecuzione di script basata sul tipo di file o sul metodo; fornisce script CGI in base al tipo di media o al metodo della richiesta.
mod_alias	Crea degli alias e reindirizza; fornisce la mappatura delle diverse parti del file system host nella struttura ad albero dei documenti, e il reindirizzamento degli URL.
mod_auth	Autenticazione dell'utente tramite l'autenticazione con file di testo, Basic e Digest.
mod_auth_anon	Autenticazione dell'utente anonimo, nello stile di FTP.
mod_autoindex	Elencazioni automatiche delle directory.
mod_cgi	Esecuzione di script CGI; elabora qualunque file con il tipo MIME application/x-httpd-cgi.
mod_define	Abilita la direttiva Define, la quale definisce una variabile che può essere espansa su qualsiasi linea di configurazione. La direttiva Define presenta lo stato Extension, ovvero, per default non viene compilata nel server.
mod_digest	Autenticazione MD5; offre l'autenticazione dell'utente con il metodo di autenticazione Digest MD5.
mod_dir	Gestione di base delle directory; si occupa dei reindirizzamenti "con barra finale" e di fornire file di indice delle directory.
mod_dms	Consente di controllare le prestazioni dei componenti di un sito con il Dynamic Monitoring Service di Oracle.
mod_env	Passaggio di ambienti agli script CGI; si occupa del passaggio di variabili ambientali agli script CGI/SSI.
mod_expires	Applica intestazioni Expires alle risorse; si occupa della generazione di intestazioni Expires in base ai criteri specificati dall'utente.
mod_fastcgi	Questo modulo di terzi supporta il protocollo fastcgi, il quale permette di mantenere un gruppo di server in fase di esecuzione per le applicazioni CGI (eliminando così il sovraccarico dovuto ai processi di avvio e inizializzazione).
mod_headers	Aggiunge intestazioni HTTP arbitrarie alle risorse; le intestazioni dei documenti possono essere unite, sostituite o rimosse.
mod_include	Documenti analizzati dal server; si occupa dei documenti HTML analizzati dal server.
mod_info	Riepiloga l'intera configurazione del server comprese le impostazioni di tutti i moduli e direttive installati.

---

mod_log_config	Modulo di logging configurabile dall'utente sostitutivo di mod_log_common.
mod_log_common	Si occupa delle richieste di logging rivolte al server con il Common Log Format o con un formato specificato dall'utente.
mod_mime	Determina e registra i tipi di documenti con le estensioni di file.
mod_negotiation	Negoziazione del contenuto.
mod_oc4j	Instrada le richieste da Oracle HTTP Server ai Oracle9iAS Container for J2EE (OC4J), fornendo il protocollo ajp13 usato per le comunicazioni con il servlet engine.
mod_oradav	Il modulo Oracle (un'applicazione OCI scritta con C) è un'implementazione ampliata di mod_dav, ed è integrato con Oracle HTTP Server. mod_oradav svolge attività di lettura/scrittura nei file locali e nei database Oracle.
mod_oss1	Imposta i parametri di sicurezza per mod_oc4j. Questo modulo Oracle abilita una crittografia robusta per il server HTTP.
mod_osso	Abilita single sign-on per Oracle HTTP Server. mod_osso esamina le richieste in ingresso e stabilisce se le risorse richieste sono protette; in caso affermativo, recupera i cookie del server HTTP per l'utente.
libproxy (mod_proxy)	Capacità proxy con caching; fornisce un proxy server con caching HTTP 1.0.
mod_rewrite	Potente funzione di mappatura dei nomi di URL con file che utilizza le espressioni regolari; fornisce un motore di riscrittura basato su regole per riscrivere gli URL richiesti al momento.
mod_setenvif	Imposta le variabili ambientali in base alle informazioni del client; offre la possibilità di impostare le variabili ambientali in base agli attributi della richiesta.
mod_so	Supporto per il caricamento dei moduli al momento dell'esecuzione; si occupa del caricamento del codice e dei moduli eseguibili nel server al momento dell'avvio o del riavvio.
mod_speling	Corregge automaticamente gli errori di ortografia minori negli URL; cerca di correggere gli errori di digitazione negli URL inseriti dagli utenti, ignorando le maiuscole e minuscole e consentendo al massimo un errore.
mod_status	Visualizzazione dello stato del server; consente a un Webmaster, a un DBA o ad un amministratore di server di scoprire le prestazioni del server, presentando una pagina HTML che fornisce le statistiche aggiornate sul server in un formato di facile lettura.
mod_unique_id	Crea un ID univoco per ogni richiesta.
mod_userdir	Home directory degli utenti; si occupa delle directory specifiche degli utenti.
mod_usertrack	Riconoscimento degli utenti tramite cookie.
mod_vhost_alias	Abilita l'hosting virtuale di massa configurato in modo dinamico.

## Communication Services e IIS

Le richieste in ingresso sono gestite dai Communication Services di Oracle, noti più comunemente come *Web Server*. Oracle9iAS viene fornito con il Web Server Apache e supporta anche Microsoft Internet Information Server (IIS). Con l'Oracle Plug-in for Microsoft IIS è possibile richiamare direttamente oggetti PL/SQL e Java memorizzati su Web da un database Oracle. L'Oracle Plug-in for Microsoft IIS offre funzionalità in un ambiente Microsoft IIS simile ai moduli per Oracle HTTP Server: *mod\_plsql* e *mod\_ose*. Il suo impiego consente di accedere ai componenti delle applicazioni Web in uno dei modi seguenti:

- Passando un prefisso di directory virtuale preconfigurato (accesso PL/SQL).
- Passando un'estensione di file predefinita e dei prefissi di directory virtuali, memorizzati nel file di configurazione Java (accesso Java).

### Accesso ai componenti Web PL/SQL

L'Oracle Plug-in for Microsoft IIS supporta l'accesso a pagine server PL/SQL e a stored procedure PL/SQL scritte con Oracle PL/SQL Web Toolkit. Se si ha già un'applicazione PL/SQL, non dovrebbe essere un problema richiamarla da IIS. Le richieste PL/SQL vengono filtrate con un prefisso predefinito e vengono eseguite con connessioni al database comuni. Gli sviluppatori sono in grado di configurare l'accesso a componenti PL/SQL da più database.

### Accesso ai componenti Web Java

L'Oracle Plug-in for Microsoft IIS supporta l'accesso a JSP e alle servlet. Le richieste Java vengono filtrate con un file di configurazione Java.

Per ulteriori informazioni, si faccia riferimento a *Oracle Plug-in for Microsoft IIS Configuration and User's Guide* nella Oracle9i Application Server Documentation Library.

## 39.2 Content Management Services

I *Content Management Services* fanno in modo che tutto il contenuto, a prescindere dal tipo di file, sia accessibile in un'unica gerarchia di file eterogenea tramite browser Web, una rete Microsoft Windows, il protocollo File Transfer Protocol (FTP) o un client di posta elettronica. È inoltre possibile utilizzare questi servizi per configurare delle capacità di ricerca file sofisticate, avvisi di eventi e funzionalità di check-in e check-out a supporto dei progetti in collaborazione. Sui server basati su Windows NT o Windows 2000, molti di questi servizi sono disponibili tramite Microsoft IIS. Per abilitare questi stessi servizi su tutti i sistemi operativi, Oracle ha introdotto l'Internet File System.

### Oracle Internet File System

L'Internet File System (IFS) di Oracle è un servizio che memorizza i file in un database Oracle. Dal punto di vista dell'utente, Oracle IFS appare come un qualunque altro file system accessibile tramite browser Web, Esplora risorse di Microsoft, FTP, NFS o un client di posta elettronica. Il fatto che i file siano memorizzati nel database risulta trasparente agli utenti, perché essi non interagiscono direttamente con quest'ultimo.

**NOTA** La recentissima 9iAS Release 2 iFS deve utilizzare un database Oracle9i, mentre la Release 1 può usare Oracle8i o Oracle9i.

A differenza di altri file system, Oracle IFS memorizza tutti i file di documento nello stesso file system. Per esempio, si possono visualizzare file di messaggi di posta elettronica e rubriche, file Web e file di elaboratore di testi o di foglio di calcolo in una singola gerarchia di file, oppure è possibile effettuare una singola ricerca per individuare tutti i riferimenti a un argomento.

### Gestione del file system

Come avviene con la maggior parte dei file system tradizionali legati a un sistema operativo, è possibile gestire i propri file o quelli per i quali si dispone delle autorizzazioni adeguate. Ecco alcune delle principali caratteristiche di gestione dei file:

- **Rinomina dei file** Si possono rinominare i file che si possiedono.
- **Eliminazione di file** Si possono eliminare i file per i quali si dispone delle autorizzazioni corrette.
- **Caricamento di file** Si possono caricare file con FTP, NFS, Esplora risorse (utilizzando Server Message Block e Cartelle Web) e il Web.
- **Impostazione della scadenza automatica dei file** È possibile impostare una data di scadenza per ciascun file caricato su Oracle IFS. Il file viene automaticamente cancellato in quella data.
- **Modifica degli attributi dei file** È possibile visualizzare e modificare gli attributi dei file, come la descrizione, il nome e le autorizzazioni per il controllo degli accessi.

A differenza della maggior parte dei file system tradizionali legati a un sistema operativo, IFS è concepito per consentire lo sviluppo e la gestione di gruppo collaborativa dei documenti. Queste caratteristiche aggiuntive comprendono.

- **Check-in, check-out (CICO)** Per i progetti di gruppo, è possibile estrarre i file (check out), in modo che gli altri non possano sovrascrivere il proprio lavoro. I file rimangono bloccati fino a quando non vengono reinseriti (check in) o un amministratore rilascia il blocco.
- **Controllo delle versioni** Gli sviluppatori possono decidere di fare in modo che un file abbia un controllo delle versioni. Ogni volta che il file con controllo delle versioni viene reinserito (check in), viene creata e memorizzata una nuova versione. Le versioni dei file possono essere conservate o eliminate secondo necessità.
- **Ricerca** Oracle IFS include una utility Find che usa le estensioni Oracle Text per ricercare in più tipi di file. Questa utility di ricerca avanzata permette di cercare rapidamente senza conoscere la sintassi del database.
- **Più cartelle per file** Questa funzione consente di assegnare più cartelle ai file, evitando di fare più copie del file quando lo stesso documento rientra in diverse categorie. Questa funzione riduce il lavoro di manutenzione quando i file vengono aggiornati e contribuisce a risparmiare spazio di memorizzazione.
- **Attributi dei file estensibili** Gli sviluppatori di applicazioni possono estendere Oracle IFS in modo che gestisca nuovi tipi di file per informazioni personalizzate. Per ogni nuovo tipo è possibile definire degli attributi personalizzati sull'informazione, che sono poi usati per tenere traccia e ricercare l'informazione. Questo è utile in particolare per rintracciare tipi di file binari o proprietari di un'applicazione che tradizionalmente sono difficili da cercare.

### 39.3 Ultra Search

La funzione *Ultra Search* di Oracle è nuova nella release del database Oracle9i e permette di effettuare ricerche all'interno di database, nonché in pagine HTML statiche. Gli altri motori di

ricerca non possono vedere il contenuto all'interno di un database e non sarebbero in grado di trovare, per esempio, documenti, articoli di giornale e simili memorizzati in un database. Ultra Search unifica le aree di ricerca tra repository aziendali eterogenei, siti Web e contenuto groupware. Oracle Ultra Search include un'interfaccia Web, Web crawling e strumenti di amministrazione della ricerca per offrire un'interfaccia unificata per le applicazioni di ricerca enterprise e per portali verticali.

## 39.4 Business Logic Services

I *Business Logic Services* sono gli strumenti e i linguaggi di sviluppo che supportano la logica delle proprie applicazioni. Con i Business Logic Services è possibile realizzare ed eseguire applicazioni Web personalizzate. I paragrafi seguenti descrivono gli elementi principali che i Business Logic Services forniscono in Oracle9iAS.

### Componenti business per Java di Oracle

I *componenti business* per Java di Oracle sono una struttura Java e XML pura al cento per cento che consente lo sviluppo produttivo, la distribuzione portabile e la personalizzazione flessibile di applicazioni per database a più livelli a partire da componenti business riutilizzabili. Gli sviluppatori di applicazioni possono usare questa struttura per:

- creare e verificare la logica business in componenti che si integrano automaticamente con i database;
- riutilizzare la logica business attraverso più viste SQL dei dati che supportano diversi task dell'applicazione;
- accedere e aggiornare le viste a partire da servlet, pagine JSP e da client Thin-Java Swing;
- personalizzare la funzionalità dell'applicazione in strati senza modificare l'applicazione prodotta.

Per un'introduzione a Java, fare riferimento al Capitolo 34.

### Oracle PL/SQL

Oracle *PL/SQL* è un motore scalabile per l'applicazione della logica business ai dati contenuti nella Oracle Database Cache e in database Oracle. Esso consente di richiamare procedure PL/SQL memorizzate nei database Oracle attraverso i propri browser Web. Le stored procedure recuperano i dati da tabelle del database e generano pagine HTML, che includono i dati che vengono restituiti al browser del client. Per un'introduzione a PL/SQL, fare riferimento al Capitolo 27.

### Oracle Forms Services

Se è stata installata l'Enterprise Edition di Oracle9iAS, si potranno eseguire applicazioni basate sulla tecnologia Oracle Forms sia su Internet sia sulla intranet aziendale. A livello di application server, gli Oracle *Forms Services* sono costituiti da un listener e da un motore runtime, nel quale è memorizzata la logica dell'applicazione. A livello del client, gli Oracle Forms Services sono

invece costituiti da un'applet Java, che fornisce l'interfaccia utente per il motore runtime, e da Oracle JInitiator (un plug-in Java), che fornisce la capacità di specificare l'uso di una specifica Java virtual machine sul client.

In Oracle9i, quando si invia un URL per avviare un'applicazione basata su Oracle Forms, il listener Web accetta la richiesta e scarica l'applet Oracle Forms nel browser. Questa applet stabilisce quindi una connessione permanente con un motore runtime Oracle Forms. Tutta l'elaborazione avviene tra l'applet Oracle Forms e il motore runtime Oracle Forms Services, che gestisce senza interruzioni qualsiasi query o la trasferisce al database.

## 39.5 Presentation Services

I *Presentation Services* forniscono contenuto dinamico ai browser dei client, grazie al loro supporto per servlet, Java Server Pages, script Perl/CGI, PL/SQL Server Pages, moduli e business intelligence. I *Presentation Services* possono servire per costruire lo strato di presentazione per le applicazioni Web. Sebbene Oracle collochi questi servizi in un gruppo separato, potete considerarli raggruppati con i Business Logic Services, per il fatto che la logica dell'applicazione può risiedere in questi componenti. Esiste la possibilità di suddividere la logica business in componenti eseguibili, che possono poi essere chiamati o invocati da questi servizi.

### Apache JServ

*Apache JServ* è un servlet engine Java puro al cento percento, totalmente compatibile con le seguenti specifiche Java di Sun Microsystems:

- Java Servlet APIs versione 2.0.
- Java Runtime Environment (JRE) versione 1.1.

A partire dalla versione 1.0.2.2 di Oracle9iAS, Oracle offre il servlet engine J2EE. Apache JServ funziona con qualunque Java Virtual Machine compatibile con questo JRE ed esegue qualunque servlet Java compatibile con questa versione della specifica Java Servlet APIs. Quando il server HTTP riceve una richiesta relativa a una servlet, essa viene instradata a mod\_jserv, che inoltra la richiesta al servlet engine Apache JServ.

### OracleJSP (JavaServer Pages)

La tecnologia *JSP* estende la tecnologia Java Servlet e consente l'uso di chiamate e scriptlet Java nelle pagine HTML e XML. Grazie alle pagine JSP è possibile creare interfacce utente combinando dati di tipo statico e contenuto dinamico. Le pagine JSP supportano lo sviluppo a componenti, separando la logica business (in genere in chiamate a stored procedure JavaBeans o PL/SQL) dalla presentazione, il che permette di concentrarsi sulle proprie aree di competenza. Di conseguenza, gli sviluppatori JSP (che potrebbero non conoscere Java) possono concentrarsi sulla logica della presentazione, mentre gli sviluppatori Java possono occuparsi della logica business. Per informazioni generali sulle JSP, fare riferimento alla specifica JavaServer Pages (disponibile al sito <http://java.sun.com>).

OracleJSP è un'implementazione di JavaServer Pages versione 1.1 (insieme ad alcune caratteristiche di HSP 1.2) come specificato da Sun Microsystems, e questa specifica viene tuttavia estesa per offrire i vantaggi seguenti.

■ **Portabilità tra diversi ambienti servlet** Le pagine OracleJSP sono accettate da tutti i server Web che supportano le servlet Java realizzate in base alla versione 2.0 o successive della specifica. Di conseguenza, sarà possibile migrare a Oracle9i Application Server le pagine JSP esistenti compatibili con la versione 1.0 della specifica senza doverle riscrivere.

■ **Supporto per SQLJ** *SQLJ* è una sintassi standard per incorporare comandi SQL direttamente nel codice Java. OracleJSP supporta la programmazione SQLJ delle scriptlet JSP. Questo include il supporto per un'estensione di file aggiuntiva, *sqljsp*, che fa sì che il programma di conversione OracleJSP invochi il programma di conversione Oracle SQLJ. Per esempi sulla programmazione SQLJ, fare riferimento al Capitolo 35.

■ **Oracle JSP Markup Language (JML)** Oracle offre l'insieme delle tag JML come libreria di tag di esempio, che permette agli sviluppatori che non conoscono bene la sintassi Java di avere una spinta iniziale nell'uso di cicli, espressioni condizionali e altri costrutti di programmazione di alto livello.

■ **National Language Support (NLS) esteso** OracleJSP offre un supporto esteso per NLS per gli ambienti servlet che non sono in grado di codificare i parametri di richiesta a più byte e le impostazioni delle proprietà dei bean. Per tali ambienti, OracleJSP offre il parametro di configurazione *translate\_params*, che può essere attivato per indicare a OracleJSP di sostituire il contenitore delle servlet e procedere esso stesso alla codifica.

■ **Tipi di dati estesi** OracleJSP fornisce le classi JavaBean *JmlBoolean*, *JmlNumber*, *JmlFPNumber* e *JmlString* nel package *oracle.jsp.jml* per sottoporre a wrapping i tipi di dati Java più comuni. Questi tipi di dati estesi forniscono un mezzo per aggirare le limitazioni dei tipi primitivi e delle classi wrapper di Java nel package standard *java.lang*.

■ **JavaBeans personalizzati** OracleJSP include un gruppo di JavaBeans personalizzati per accedere rapidamente al database Oracle.

A partire da Oracle9iAS Release 2, il nuovo nome di OracleJSP sarà “OC4J JSP Container,” che unisce la vecchia tecnologia OracleJSP con Orion di Ironflare.

## Oracle PL/SQL Server Pages (PSP)

Le PSP di Oracle sono una delle più interessanti tra le nuove caratteristiche offerte agli sviluppatori Oracle PL/SQL tradizionali in Oracle9iAS. Le pagine PSP sono analoghe alle JavaServer Pages, con la differenza che utilizzano PL/SQL, invece di Java, per lo scripting sul lato server. A differenza delle altre tecnologie per pagine server, le pagine PSP sono componenti completamente compilati che vengono eseguiti come stored procedure di Oracle. Il servizio Oracle PSP include il PSP Compiler e il PL/SQL Web Toolkit.

Utilizzando questo servizio quando si sviluppano le applicazioni, è possibile separare la formattazione della pagina dalla logica dell'applicazione. Partendo da una pagina Web o da una stored procedure già esistente, è possibile creare pagine Web dinamiche incorporando delle tag PL/SQL. Queste pagine Web dinamiche sono in grado di svolgere operazioni con il database e di visualizzare i risultati come codice HTML, XML o come semplice testo. Normalmente, è previsto che una pagina per server PL/SQL venga visualizzata in un browser Web. Inoltre, può essere recuperata e interpretata da un programma in grado di fare richieste HTTP, come un'applicazione Java o Perl, consentendo di migliorare lo sviluppo modulare e il riutilizzo dell'applicazione.

Se si pensa di usare le pagine PSP come soluzione per lo strato di presentazione, è importante comprendere che, a meno che si utilizzi un Database Cache Server, il codice PL/SQL verrà eseguito sul database server, non sull'application server. Java e la maggior parte degli altri linguaggi eseguono il proprio codice sull'application server, rendendo scalabili queste soluzioni senza usare un Database Cache Server.

## Interprete Perl

L'*Interprete Perl* è un ambiente runtime Perl permanente che, essendo incorporato nel server HTTP Apache, evita il sovraccarico legato all'avvio di un interprete esterno, come avviene in una tradizionale chiamata CGI. Quando l'Oracle HTTP Server riceve una richiesta Perl, essa viene instradata attraverso mod\_perl all'interprete Perl, che si occupa della sua elaborazione.

## 39.6 Business Intelligence Services

I *Business Intelligence Services* di Oracle permettono di gestire e analizzare quotidianamente l'azienda e il sito Web, consentendo la distribuzione e la condivisione della business intelligence sul Web o sulla intranet aziendale.

### Oracle Reports Services

Grazie ai *Reports Services* di Oracle e ai suoi *Reports Servlet Services*, è possibile creare ed eseguire report Oracle nuovi o già esistenti sulla intranet di una società, sulla extranet di una società esterna oppure pubblicarli su Internet. Gli Oracle Reports Services sono ottimizzati per distribuire le applicazioni Oracle Reports (report e grafici) in un ambiente a più livelli. I Reports Services sono costituiti dal componente server, dai motori runtime e dal motore di esecuzione del servlet. In Oracle9iAS, quando un client inoltra una richiesta di creazione di un report, il listener Web dell'Oracle HTTP Server la instrada al componente del server Reports Services. Il server instrada la richiesta al motore runtime di Oracle Reports Services, il quale esegue il report. L'output del report viene quindi inviato al client tramite il listener Web dell'Oracle HTTP Server. I report possono essere formattati come HTML, Adobe Acrobat o come semplice testo a beneficio dell'utente. Inoltre, gli sviluppatori possono personalizzare facilmente il report in modo da importarlo in Microsoft Excel o in qualsiasi altro tipo di documento comunemente supportato, come Multipurpose Internet Mail Extensions (MIME).

### Oracle Discoverer 4i Viewer/Oracle9iAS Discoverer

Oracle9iAS Discoverer (noto precedentemente come *Discoverer 4i Viewer*) è uno strumento per l'esecuzione e la visualizzazione di cartelle di lavoro (report) create con Oracle Discoverer 4i Plus sul Web. Rivolto a utenti esperti, Discoverer permette di accedere alle informazioni del database e di incorporarle nel sito utilizzando strumenti con interfaccia grafica (GUI) e una visualizzazione dei dati WYSIWYG (What You See Is What You Get) senza essere sviluppatori SQL esperti. È possibile pubblicare dei report attivi sui siti Web creando un URL collegabile che indica a Discoverer quali cartelle di lavoro aprire. Un clic sull'URL invoca la query della cartella di lavoro relativa al database e restituisce dei risultati attivi al browser. Gli utenti interagiscono con i risultati della query per comprimere o espandere, per immettere valori in parametri facoltativi, o per seguire i collegamenti ad altre cartelle di lavoro o applicazioni.

## 39.7 Portal Services

I *Portal Services* possono essere utilizzati per costruire portali che integrano tutto il contenuto in un'unica pagina Web. I portali offrono agli utenti una singola visualizzazione centralizzata e

personalizzata dei dati e delle applicazioni rilevanti. Con i Portal Services e Portal-to-Go di Oracle9iAS, è possibile rendere accessibili i portali a client fissi e mobili. Sebbene sia anche possibile costruire semplici applicazioni nell'ambito di Oracle Portal, la sua finalità principale è offrire applicazioni agli utenti tramite un modello a sottoscrizione.

## Oracle Portal

Un *portale enterprise* è un'applicazione Web che fornisce un punto di ingresso comune e integrato per accedere a tipi di dati dissimili su una singola pagina Web. Per esempio, si possono creare portali che offrono agli utenti la possibilità di accedere ad applicazioni Web, a documenti aziendali, a report di business intelligence, a grafici e a collegamenti sia interni sia esterni alla intranet aziendale. È possibile gestire l'esperienza degli utenti, creando e amministrando pagine di portale che contengono portlet. Dal punto di vista degli utenti, i *portlet* di una pagina appaiono come regioni di una pagina HTML che presentano testo o immagini come collegamenti a ulteriori informazioni o applicazioni. I portlet forniscono accesso a risorse Web come applicazioni, pagine Web o raccolte di contenuti raggruppati. I portlet sono di proprietà dei *portlet provider*, che forniscono il collegamento di comunicazione tra la pagina del portale e i portlet. Quando un'applicazione è pronta per l'utente, viene *pubblicata*, e l'amministratore del portale la mette a disposizione del pubblico per cui è concepita. A questo punto, gli utenti effettuano una *sottoscrizione* al portlet, collocandolo effettivamente sulle proprie pagine personalizzate.

## Portal-to-Go

*Portal-to-Go* è uno strato di traslazione disponibile sull'Enterprise Edition del database che consente la visualizzazione del contenuto Web su dispositivi WAP (Wireless Application Protocol), come i telefoni cellulari dotati di browser e i computer palmari. *Portal-to-Go* isola l'acquisizione del contenuto dalla sua consegna. Funziona come un foglio di stile XSLT, offrendo uno strato di formato intermedio, *Portal-to-Go XML*, tra il formato di origine e quello di destinazione. *Portal-to-Go XML* è un insieme di DTD (Document Type Definitions) e convenzioni per documenti XML usato per definire il contenuto e gli oggetti interni in Oracle *Portal-to-Go*.

## Dynamic Services di 9i

I *Dynamic Services* utilizzano XML, XSLT e l'API di un client Java o PL/SQL comune per trasformare qualunque contenuto Web o applicazione tradizionale accessibile tramite Web in un servizio utilizzabile da altre applicazioni o dall'utente finale. I servizi sono chiamate ad applicazioni o origini dati Web sottoposte a wrapping in un package di servizi XML che definisce le API di input e output, le proprietà dell'origine dati e il flusso di esecuzione. Considerando ciascun servizio dinamico come un componente di applicazione, i servizi possono essere raggruppati e pubblicati insieme per formare un'interfaccia per applicazione Web coerente. I *Dynamic Services* offrono supporto per la maggior parte dei protocolli Web (per esempio, HTTP, UDDI, LDAP, SMTP, XML Schema o SOAP) e consentono agli sviluppatori di eseguire il wrapping di applicazioni legacy con API personalizzate, dando loro una nuova interfaccia basata su standard. Come parte dell'architettura dei *Dynamic Services*, Oracle fornisce un registro centrale per la gestione e la definizione delle proprietà, dei diritti, della sintassi e del flusso di esecuzione delle applicazioni contenute all'interno dei *Dynamic Services*. Oracle fornisce inoltre un *Dynamic Services Adapter* per incorporare la gestione dei *Dynamic Services* nel registro di *9iAS Portal*, in modo

che qualunque servizio definito in Dynamic Services sia immediatamente disponibile per la pubblicazione come portlet.

## 39.8 Kit per sviluppatori

A supporto dello sviluppo e della distribuzione delle applicazioni, Oracle9iAS fornisce diversi kit contenenti librerie e strumenti. Con questi *kit per sviluppatori* è possibile realizzare applicazioni Web in grado di accedere al proprio database.

### Il kit Java Messaging Service (JMS) di Oracle

Il Java Messaging Service (JMS) di Oracle estende la versione 1.02 della specifica standard di Java Message Service pubblicata da Sun Microsystems. Oltre alle caratteristiche standard di JMS, Oracle JMS fornisce un'API Java per Oracle Advanced Queuing (AQ). Questa API supporta le operazioni amministrative AQ e altre caratteristiche AQ tra cui:

- **Un'API amministrativa** per creare tabelle di code, code e argomenti.
- **Comunicazione punto-multipunto**, che estende la comunicazione punto-punto standard utilizzando gli elenchi di destinatari per argomenti.
- **Propagazione dei messaggi tra destinazioni**, che consente alle applicazioni di definire sottoscrittori remoti.
- **Sessioni transazionali**, in modo da poter eseguire operazioni JMS, nonché SQL, in una transazione comune.
- **Conservazione dei messaggi** dopo che i messaggi sono stati rimossi dalla coda.
- **Ritardo dei messaggi**, per rendere i messaggi visibili dopo un tempo specificato.
- **Gestione delle eccezioni**, in modo che i messaggi possano essere spostati in code di eccezioni se non possono essere elaborati con successo.
- Tipi di messaggi **AdtMessages di Oracle8i o Oracle9i**, che sono memorizzati nel database come oggetti Oracle. Di conseguenza, il payload del messaggio può essere interrogato dopo essere stato accodato. Le sottoscrizioni possono essere definite sul contenuto di questi messaggi, oltre alle proprietà dei messaggi.

### Oracle SQLJ

Oracle *SQLJ* è un preprocessore che gli sviluppatori possono usare per incorporare operazioni SQL statiche nel codice Java. Un *programma SQLJ* è un metodo Java che contiene istruzioni SQL statiche incorporate conformi alla sintassi ANSI-standard di SQLJ Language Reference. Le operazioni SQL statiche sono predefinite. Le operazioni stesse non cambiano in tempo reale mentre un utente esegue le applicazioni, sebbene i valori restituiti possano mutare in modo dinamico.

Oracle SQLJ è costituito da un programma di conversione e da un componente runtime. Il *programma di conversione* converte le istruzioni SQL incorporate in chiamate al *componente runtime SQLJ*, che esegue le operazioni SQL. Con SQLJ standard questo normalmente avviene tramite chiamate a un driver JDBC. Nel caso di Oracle9iAS si utilizza un driver JDBC Oracle. Quando si eseguono le applicazioni SQLJ, il componente runtime viene invocato per gestire le operazioni SQL. Per ulteriori informazioni su SQLJ, fare riferimento al Capitolo 35.

## Kit di sviluppo XML di Oracle

L'Extensible Markup Language (XML) si sta rapidamente affermando come il nuovo standard per lo sviluppo Web, in particolare tra i fornitori di contenuto e servizi business-to-business (B2B). L'Oracle XML Developer's Kit (XDK) contiene delle librerie di componenti XML e delle utility utilizzabili per abilitare applicazioni e siti Web a XML. Utilizzando le stored procedure Java, l'API per chiamare i componenti è disponibile sia per le applicazioni Java sia per quelle PL/SQL. Per ulteriori informazioni su XML, fare riferimento al Capitolo 41.

Oracle9i è dotato di diverse operazioni di database avanzate per la memorizzazione di XML tramite SQL e il rendering dei dati tradizionali di database come XML. Queste funzionalità sono necessarie per il supporto dell'e-business B2B e business-to-customer (B2C), per le applicazioni package e per la gestione del contenuto Internet. L'area principale di supporto XML in Oracle9i è da rintracciare negli XDK XML incorporati.

Grazie alla presenza di Java precaricato e del XDK C collegato in Oracle9i, gli sviluppatori possono accedere facilmente alle funzionalità del World Wide Web Consortium (W3C) che generano, manipolano, rendono e memorizzano i dati formattati con XML in Oracle9i. Disponibili anche in PL/SQL e C++, gli XDK offrono parser XML/XSLT, processori XML Schema, XML Class Generators, XML Transviewer Beans e XSQL Servlet, fornendo in tal modo le caratteristiche fondamentali di sviluppo che consentono agli sviluppatori di abilitare rapidamente le loro applicazioni a XML.

## Kit di sviluppo LDAP di Oracle

Il *Lightweight Directory Access Protocol* (LDAP) Developer's Kit supporta la connettività dei client con Oracle Internet Directory, il server delle directory LDAP di Oracle. Oracle Internet Directory unisce un'implementazione nativa dello standard LDAPv3 dell'Internet Engineering Task Force (IETF) con un data store backend di Oracle8i o Oracle9i. In particolare, si può usare l'Oracle LDAP Developer's Kit per lo sviluppo e il monitoraggio di applicazioni abilitate a LDAP. Esso supporta connessioni cifrate e chiamate client ai servizi di directory, quindi può servire per gestire i dati delle directory.

## 39.9 Caching Services

Per poter contare su una certa scalabilità del proprio sito Web di e-business, Oracle9iAS fornisce i *Caching Services* nella Enterprise Edition della suite. Questi servizi includono Oracle Web Cache e, in 9iAS Release 1, Oracle Database Cache. Il primo è un servizio mirato al contenuto che migliora le prestazioni, la scalabilità e la disponibilità dei siti Web attraverso un caching di pagine sia statiche sia dinamiche. Oracle Database Cache, che non è disponibile in Oracle9iAS Release 2, è invece una cache per database di livello intermedio che riduce il carico tra i livelli dell'applicazione e del database attraverso il caching dei dati richiesti di frequente, evitando inutili viaggi di andata e ritorno sulla rete per ottenere dei dati di sola lettura.

### Oracle Database Cache

La *Database Cache* di Oracle si colloca sul livello intermedio sotto forma di servizio di 9iAS. Essa migliora le prestazioni e la scalabilità delle proprietà che accedono ai database Oracle attraverso il caching dei dati usati di frequente sulla macchina di livello intermedio. Grazie alla

Oracle Database Cache, le applicazioni hanno una capacità di elaborazione delle richieste che è di diverse volte superiore alla loro capacità originale, in quanto le connessioni con il database back-end sono notevolmente ridotte. Il servizio Oracle Database Cache supporta l'esecuzione di servlet con conservazione dello stato, JavaServer Pages, Enterprise JavaBeans e oggetti CORBA nella JVM di Oracle8i e Oracle9i.

### **Perché usare la Oracle Database Cache?**

Se le applicazioni soddisfano i criteri seguenti, si può utilizzare la Oracle Database Cache per migliorare la scalabilità dei siti Web e le prestazioni delle applicazioni.

- Le applicazioni accedono a un database Oracle.
- Si utilizza un ambiente a più livelli, dove i client, l'Application Server e i database server Oracle sono collocati su macchine separate.
- Le applicazioni comunicano con un database Oracle attraverso l'Oracle Call Interface (OCI) o attraverso uno strato di accesso costruito su OCI, come JDBC-OCI, ODBC o OLE.
- Nella maggior parte dei casi, gli utenti del sito Web o delle applicazioni generano query di sola lettura.
- Le applicazioni possono tollerare un certo ritardo di sincronizzazione con i dati del database di origine. Ciò significa che la cache non deve necessariamente essere aggiornata come i dati in tempo reale del database master (è necessario decidere con quale frequenza aggiornare i dati).

### **L'ambiente della Database Cache di Oracle**

Nell'*ambiente della Oracle Database Cache*, il software della cache è costituito da una istanza di database thin di livello intermedio per il caching dei dati con accesso frequente e l'esecuzione di un software intelligente che instrada le query. Il database di origine è in esecuzione sul proprio livello backend, e rappresenta la memorizzazione originale e primaria dei dati. Attualmente, il servizio Oracle Database Cache è in grado di effettuare il caching dei dati provenienti da un solo database di origine.

### **Esempio di gestione delle richieste da parte della Oracle Database Cache**

Quando gli utenti richiedono dei dati ad accesso frequente, la richiesta passa dal client alla Oracle Database Cache attraverso il Web Server, e la cache del database restituisce i dati. Dal momento che i dati sono memorizzati sullo stesso livello, essi vengono restituiti rapidamente e la richiesta non deve passare al server del database di origine per il reperimento dei dati. Se un nuovo utente richiede le medesime informazioni, la richiesta viene ancora una volta soddisfatta nella cache di database di livello intermedio, che restituisce i dati molto rapidamente. Se i dati non sono presenti nella cache, la richiesta viene allora instradata verso il database di origine per la risoluzione, memorizzata nella cache di livello intermedio, e servita all'utente. Qualunque richiesta successiva degli stessi dati verrà soddisfatta dalla cache.

### **Differenze tra la Oracle Database Cache e il database di origine**

Sebbene la Oracle Database Cache assomigli a una normale istanza di database collocata al livello intermedio, esistono importanti differenze.

- La Oracle Database Cache memorizza i dati in una cache, ma non è una forma di memorizzazione permanente.
- Non consente operazioni di backup e di ripristino. Se si deve effettuare un backup dei dati, li si dovrà memorizzare nel database di origine.
- La Oracle Database Cache non può essere usata per creare tabelle e altri oggetti di database. L'unico modo per collocare dei dati in questo ambiente è il caching dei dati dal database di origine nell'istanza di livello intermedio.

- La Oracle Database Cache non fornisce garanzie di recupero trasparente in caso di problemi (TAF, Transparent Application Fail-over), perché la Oracle Database Cache non è una vera area per la memorizzazione permanente dei dati; è soltanto una versione speculare dell'istanza sorgente.
- La Oracle Database Cache non può essere trattata come un database che partecipa alle transazioni globali per evitare inutili e pesanti commit a due fasi.

### **In che modo le applicazioni sfruttano la Oracle Database Cache ?**

Per sfruttare i vantaggi della Oracle Database Cache, l'unica cosa da fare è configurare l'ambiente delle applicazioni. Non si dovranno modificare le applicazioni se queste utilizzano istruzioni SQL per accedere al database, o nel caso in cui siano collegate con OCI tramite librerie dinamiche e siano direttamente stratificate su OCI. Se l'applicazione soddisfa uno di questi due criteri, le query vengono instradate automaticamente verso la cache del database di livello intermedio.

## **Oracle Web Cache**

La *Web Cache* di Oracle è un servizio di caching del Web Server che migliora le prestazioni, la scalabilità e la disponibilità dei siti Web con molto traffico che si appoggiano a Oracle9iAS e a database Oracle. La maggior parte dei webmaster tiene delle statistiche sui propri siti e ha un'idea ben precisa di quali siano le pagine richieste con maggiore frequenza. Memorizzando le pagine con accesso frequente nella memoria virtuale, la Oracle Web Cache riduce la necessità di elaborare ripetutamente le richieste relative a quegli URL sul Web server. A differenza dei proxy server legacy e di altri Web server che gestiscono solo immagini statiche e testo, il servizio Oracle Web Cache memorizza nella cache contenuto HTTP sia statico sia generato dinamicamente da uno o più Web server per applicazioni. Grazie alla Oracle Web Cache, i client Web possono godere di un recupero più rapido delle pagine, riducendo nel contempo in modo significativo il carico sui server HTTP.

La Oracle Web Cache è collocata di fronte agli Oracle HTTP Server per memorizzare nella cache le loro pagine e per fornire il contenuto agli utenti Web che lo hanno richiesto. Quando i browser Web accedono ai siti Web, inviano delle richieste HTTP alla Oracle Web Cache, che agisce come un server virtuale o un router di richieste virtuale per gli Oracle HTTP Server. Se il contenuto richiesto è cambiato o non è più aggiornato, il servizio Oracle Web Cache recupera il nuovo contenuto da un Oracle HTTP Server.

### **Caratteristiche principali di Oracle Web Cache**

Le caratteristiche principali di Oracle Web Cache la rendono ideale per i siti Web dinamici di e-commerce che ospitano cataloghi online, servizi di notizie, servizi B2B e portali:

- **Caching del contenuto generato in modo statico e dinamico** Inserisce i documenti nella cache in base alle regole e agli orari specificati.
- **Invalidazione della cache** Supporta l'invalidazione delle pagine come strumento per mantenere coerenti le pagine della Oracle Web Cache con il contenuto presente su Oracle9iAS.
- **Controllo delle prestazioni** La misurazione incorporata delle prestazioni gestisce i problemi di prestazioni, pur mantenendo la coerenza della cache. Queste statistiche determinano un ordine di priorità del contenuto e determinano quali documenti possono essere forniti all'utente nella versione presente nella cache, e quali invece devono essere aggiornati.

- **Protezione contro il sovraccarico** Consente di impostare dei limiti al numero di richieste concorrenti passate alla Oracle Web Cache per evitare di sovraccaricarla.
- **Bilanciamento del carico** Distribuisce dinamicamente su molti Web Server le richieste che la Oracle Web Cache non è in grado di soddisfare per proprio conto. La Oracle Web Cache è progettata per gestire le richieste HTTP per un massimo di 100 server Oracle9iAS.
- **Failover di backend** Ridistribuisce automaticamente il carico sui Web server restanti nel caso in cui un server si guasti o venga scollegato. Quando il server guasto ritorna in funzione, Oracle Web Cache lo include automaticamente nel pool dei server.
- **Riconoscimento delle sessioni** Supporta i siti Web che usano il riconoscimento degli ID di sessione per tenere traccia degli utenti.
- **Sicurezza** Fornisce l'autenticazione tramite password per le attività di amministrazione, il controllo sulle porte da cui possono essere richieste le operazioni e il timeout per le connessioni inattive.

## Analisi di Oracle9iAS Clickstream

Oracle9iAS Clickstream Intelligence è uno strumento analitico e completo basato sul Web che permette di acquisire, analizzare e fare rapporti sulle interazioni Web con i clienti, i fornitori e i dipendenti. È uno dei componenti principali di Business Intelligence di Oracle9iAS Release 2. Clickstream Intelligence si appoggia agli strumenti di Business Intelligence di Oracle9iAS e al database Oracle9i per garantire una soluzione integrata ed estensibile adatta a misurare il traffico sul Web e a migliorare l'efficienza dei siti Web.

Il Runtime Administrator semplifica la configurazione, la gestione e la distribuzione delle origini di dati clickstream. Questo strumento basato sul Web consente di definire in che modo Clickstream Intelligence interpreta i dati dei file di log al Web, stabilire i tipi di dati per i quali si intende tenere traccia, configurare e popolare il database per una memorizzazione ottimale dei dati clickstream. Oracle9iAS Clickstream Intelligence Administrator's Guide descrive in dettaglio come usare il Runtime Administrator.

Oracle9iAS Clickstream Intelligence Analytics (Clickstream Analytics) mette a disposizione più di 150 report preconfigurati che visualizzano i dati acquisiti dai siti Web. I report di Clickstream Analytics, chiamati anche fogli di lavoro, sono raggruppati con altri report simili per formare cartelle di lavoro che possono essere visualizzate e alle quali si può accedere con Oracle9iAS Discoverer. Per visualizzare e analizzare i dati relativi al traffico sui siti Web, si possono utilizzare Oracle9iAS Discoverer Plus o anche Oracle9iAS Discoverer Viewer.

Oracle9iAS Clickstream Intelligence può essere installato sul database Oracle9i incluso in Oracle9i Application Server, oppure potrebbe essere installato su un database Oracle9i Enterprise Edition indipendente. Il modello di database Oracle9iAS Clickstream Intelligence viene costruito con Oracle9i Warehouse Builder, uno strumento impiegato per progettare e distribuire i database.

## 39.10 System Services

Oracle9iAS include Oracle Enterprise Manager e Oracle Advanced Security per offrire servizi di gestione del sistema e della sicurezza per le applicazioni. Questi servizi gestiscono l'ambiente Oracle e la sicurezza di rete attraverso la cifratura e l'autenticazione.

## Oracle Enterprise Manager

Il componente Oracle *Enterprise Manager* è stato migliorato al punto da non essere più un semplice strumento di amministrazione di database, ma un vero e proprio strumento di gestione del sistema che fornisce un ambiente integrato per la gestione centrale della piattaforma Oracle. Questo servizio è disponibile anche nella Enterprise Edition e combina una console GUI, Oracle Management Server, Oracle Intelligent Agent, servizi comuni e strumenti di amministrazione. In Oracle9iAS, si usa la console per gestire la Oracle Database Cache, gli Oracle Forms Services e il sistema operativo host. Dalla console è possibile svolgere le seguenti operazioni:

- Amministrare ed eseguire operazioni diagnostiche a livello centrale su Oracle Database Cache e Oracle Forms Services
- Monitorare lo stato dei prodotti Oracle e dei servizi di terzi.
- Pianificare le attività di manutenzione su più macchine secondo intervalli di tempo variabili.
- Monitorare i servizi di rete per individuare gli eventi pianificati e quelli insoliti.
- Personalizzare la visualizzazione organizzando i componenti del server in gruppi logici in relazione all'architettura unica del sistema in uso; è sempre possibile riorganizzare i servizi quando il sistema cresce e cambia, in modo da adattarli alla nuova architettura.

## Oracle Advanced Security

Il componente *Oracle Advanced Security* fornisce una suite molto completa di servizi di sicurezza per la Oracle Database Cache, la JVM di Oracle8i e Oracle9i e per PL/SQL di Oracle8i e Oracle9i. La sua funzionalità ha due facce: innanzitutto, le caratteristiche di sicurezza di rete proteggono le reti enterprise ed estendono in modo sicuro le reti aziendali verso Internet; in secondo luogo, esso integra i servizi di sicurezza e di directory, combinandoli in modo da offrire gestione degli utenti a livello enterprise e single sign on.

Le **caratteristiche di sicurezza di rete** garantiscono la riservatezza e l'integrità dei dati e l'autenticazione degli utenti tra i diversi server della rete. Inoltre forniscono agli utenti una funzione single sign on, grazie alla quale l'utente viene autenticato una volta, dopodiché l'autenticazione avviene in modo trasparente al momento delle successive connessioni ad altri database o servizi sulla rete. Grazie al single sign on, gli utenti possono accedere a più schemi e applicazioni con una singola password.

I **servizi di sicurezza e di directory** forniscono all'amministratore di sistema degli strumenti per la gestione centrale degli utenti su un servizio di directory centrale, senza dover gestire ripetutamente gli stessi utenti sui singoli database e server. Grazie a Oracle Enterprise Security Manager, uno strumento accessibile attraverso Oracle Enterprise Manager, gli utenti enterprise e le loro autorizzazioni vengono gestiti in Oracle Internet Directory o in altri servizi di directory LDAP.

### 39.11 Database Services

I *Database Services* di Oracle9i offrono dei vantaggi molto netti rispetto al database Oracle8i. Il database Oracle9i contiene molte nuove caratteristiche importanti che ottimizzano le tradizionali applicazioni aziendali e agevolano il difficile avanzamento delle applicazioni Web. Le nuove caratteristiche del database Oracle9i forniscono le prestazioni, la scalabilità e la disponibilità essenziali per le applicazioni. Queste caratteristiche, tra cui le opzioni di partizionamento, le

tabelle esterne, le query di flashback, la riorganizzazione degli oggetti in linea, la gestione dei parametri dinamici, nuove funzioni, nuovi package (come DBMS\_METADATA per l'estrazione delle DDL) e nuovi tipi di dati (come TIMESTAMP) verranno analizzate nei vari capitoli di questo libro.

## iSQL\*Plus

*iSQL\*Plus* è un'implementazione basata su browser di SQLPLUS, utilizzabile su Internet per connettersi a un Oracle RDBMS per svolgere le stesse operazioni possibili attraverso la linea di comando di SQLPLUS. L'implementazione *iSQL\*Plus* utilizza un browser Web, un Oracle HTTP Server con *iSQL\*Plus* Server e un Oracle RDBMS Server.

### 39.12 Avvio, arresto e riavvio di iAS (Apache)

Il programma di controllo di Oracle9iAS prende rispettivamente il nome di apache in NT e di apachectl in ambiente UNIX. Per avviare Oracle9iAS in NT dalla linea di comando, si verifichi che l'eseguibile apache si trovi nella variabile ambientale PATH del sistema. Per controllare se le cose stanno così, si individui l'eseguibile apache, normalmente collocato nella directory ORACLE\_HOME\Apache\Apache, e poi si verifichi il percorso selezionando l'icona di sistema nel Pannello di controllo e selezionando la scheda Environment (NT) o Avanzate (Win 2000 o Win XP Pro). Si aggiunga la directory dell'eseguibile apache nella variabile ambientale PATH, se non è impostata.

Per avviare Oracle9iAS dalla linea di comando, si inserisca il comando seguente:

```
apache -k start
```

Per arrestare Oracle9iAS dalla linea di comando, si inserisca il comando seguente:

```
apache -k shutdown
```

Inoltre, è possibile avviare e arrestare Oracle9iAs con le scelte rapide di Oracle HTTP Server in Avvio | Programmi | Oracle | Oracle HTTP Server. L'icona Servizi nel Pannello di controllo (Strumenti amministrativi/Servizi in Win XP Pro) ha una voce per Oracle HTTP Server, per dare la possibilità di avviare e arrestare Oracle9iAS da qui, oppure è possibile impostare l'avvio del servizio in modo automatico, così che Oracle HTTP Server si avvii insieme al sistema.

Per avviare Oracle9iAS in ambiente UNIX, si utilizzi il comando seguente per controllare che il percorso sia stato impostato in modo da contenere la directory in cui si trova l'eseguibile httpd:

```
echo $PATH
```

Per avviare Oracle9iAS in ambiente UNIX dalla linea di comando, si inserisca

```
apachectl
```

oppure

```
httpd
```

Per fare in modo che Oracle9iAS si avvii automaticamente quando il server viene riavviato, si modifichi il file /etc/rc.d/rc.local e si aggiunga il comando seguente:

```
$ORACLE_HOME/Apache/Apache/bin/httpd
```

Per arrestare Oracle9iAS, si inserisca

```
apachectl shutdown
```

oppure

```
kill -9 `cat $ORACLE_HOME/Apache/Apache/logs/httpd.pid`
```

Ecco la sintassi per l'uso di apache e apachectl sulla linea di comando

```
Usage: APACHE [-D nome] [-d directory] [-f file] [-n servizio]
              [-C "direttiva"] [-c "direttiva"] [-k segnale]
              [-v] [-V] [-h] [-l] [-L] [-S] [-t] [-T]
```

La Tabella 39.1 illustra le opzioni della linea di comando per UNIX e NT.

**Tabella 39.1** Opzioni della linea di comando di Apache per UNIX e NT.

OPZIONE	DESCRIZIONE
-D <i>nome</i>	Definisce un nome da usare nelle direttive <IfDefine <i>nome</i> >.
-d <i>directory</i>	Specifica una ServerRoot iniziale alternativa.
-f <i>file</i>	Specifica un ServerConfigFile alternativa.
-C "direttiva"	Elabora la direttiva prima della lettura del file di configurazione.
-c "direttiva"	Elabora la direttiva dopo la lettura del file di configurazione.
-v	Mostra il numero di versione.
-V	Mostra le impostazioni di compilazione.
-h	Elenco le opzioni per la linea di comando disponibili.
-l	Elenco i moduli compilati contenuti.
-L	Elenco le direttive di configurazione disponibili.
-S	Mostra le impostazioni analizzate (attualmente solo le impostazioni vhost).
-t	Esegue il controllo di sintassi per il file di configurazione (con controllo docroot).
-T	Esegue il controllo di sintassi per il file di configurazione (senza controllo docroot).
-n <i>nome</i>	Nome del servizio Oracle9iAS per le opzioni -k.
-k stop   shutdown	Indica all'Oracle9iAS in esecuzione di arrestarsi.

(segue)

**Tabella 39.1** Opzioni della linea di comando di Apache per UNIX e NT. (*continua*)

OPZIONE	DESCRIZIONE
-k restart	Indica all'Oracle9IAS in esecuzione di riavviarsi.
-k start	Indica a Oracle9IAS di avviarsi.
-k install   -i	Installa un servizio Oracle9IAS.
-k config	Riconfigura un servizio Oracle9IAS installato.
-k uninstall   -u	Disinstalla un servizio Oracle9IAS.



## Capitolo 40

# Guida all'amministrazione del database

- 40.1 **Creazione di un database**
- 40.2 **Avvio e arresto del database**
- 40.3 **Ridimensionamento e gestione delle aree  
di memoria per il database**
- 40.4 **Allocazione e gestione dello spazio  
per gli oggetti**
- 40.5 **Creazione e gestione dei segmenti  
di rollback**
- 40.6 **Esecuzione dei backup**
- 40.7 **Riepilogo**

In questo capitolo verranno introdotti i passaggi fondamentali coinvolti nell'amministrazione di un database Oracle. Nel lavoro di amministratore del database (DBA) rientrano diversi componenti ed esistono numerosi libri scritti appositamente per i DBA. Questo capitolo offrirà una panoramica delle attività svolte da un DBA, insieme ad alcuni consigli sull'uso degli strumenti standard per DBA.

Nei capitoli precedenti sono già state illustrate molte funzioni di un DBA, come la creazione di tabelle, indici, viste, utenti, sinonimi, database link e package. Gli argomenti trattati in questo capitolo faranno quindi riferimento alle funzioni di controllo della produzione relative al ruolo di DBA. Per ciascuno di questi argomenti, gli sviluppatori e i DBA avranno a disposizione diverse opzioni. I prossimi paragrafi di questo capitolo dovrebbero fornire informazioni sufficienti per iniziare, e per imparare quali domande rivolgere ai DBA e agli amministratori del sistema.

## 40.1 Creazione di un database

Il modo più semplice per generare uno script create database consiste nell'utilizzare Oracle Universal Installer (OUI). Quando si installa Oracle, OUI offre la possibilità di creare un database. Se si usufruisce di questa opzione, OUI creerà un database di piccole dimensioni utile per fare pratica e che potrà essere eliminato non appena conclusa questa fase di sperimentazione. Gli script create database forniti da Oracle rappresentano un modello valido per gli script create database standard. Il file build\_db.sql, contenuto nella sottodirectory /rdbms/admin nella directory home del software Oracle, offre uno script create database di esempio. Per la sintassi completa del comando, si veda la voce relativa al comando create database nel Capitolo 42.

Il comando create database viene eseguito dall'interno di SQLPLUS, tramite un account con il privilegio di sistema SYSDBA:

```
SQL> connect system/manager as sysdba  
SQL> startup nomount  
SQL> create database...
```

Per eseguire `connect as SYSDBA` correttamente, è necessario possedere le autorizzazioni adatte a livello di sistema operativo e di database. Per informazioni sui diritti di sistema operativo richiesti, fare riferimento alla documentazione Oracle specifica del sistema operativo impiegato.

## Uso di Oracle Enterprise Manager

Oracle Enterprise Manager (OEM), uno strumento con interfaccia grafica (GUI), viene fornito come parte integrante del toolset Oracle standard per consentire ai DBA di gestire i database da un PC. Il toolset OEM mette a disposizione un'interfaccia solida per l'amministrazione remota del database. Tutti i DBA possono utilizzare lo stesso repository OEM centrale (un insieme di tabelle create in un database) per svolgere il proprio lavoro. Oltre a queste modifiche, OEM include anche soluzioni per l'assegnamento e la pianificazione delle attività in modo da garantire una copertura del database ventiquattro ore su ventiquattro.

Prima di installare e configurare OEM, si dovranno prendere alcune decisioni molto importanti. Per esempio, si dovrà decidere dove creare il repository OEM, e come e quando si eseguiranno i backup per proteggere questo repository. Dato che OEM può essere impiegato come interfaccia per Oracle Recovery Manager (RMAN), le informazioni di recupero potranno essere memorizzate nel repository OEM. Esiste la possibilità di creare un piccolo database separato nel quale memorizzare il repository OEM. È consigliabile accertarsi che il backup di questo database avvenga di frequente, in modo da garantire anche il recupero del repository stesso.

Se il toolset OEM viene utilizzato da un solo DBA, non sarà necessario considerare chi gestisce l'amministrazione di database specifici nell'ambiente. Se nel sito ci sono più DBA, si renderà necessario stabilire una definizione delle attività e delle responsabilità di database, nonché il ricorso a delle pianificazioni. Con OEM è possibile garantire livelli di accesso e privilegi a ogni DBA del gruppo in base alle singole attività. Si può configurare OEM per inviare richieste e assegnamenti per posta elettronica ad altri DBA oppure per assumere il controllo di una situazione problematica allo scopo di velocizzarne la risoluzione.

Se sul sistema è presente una versione precedente di OEM, occorre procedere alla migrazione di quel repository nella versione più recente per sfruttare le nuove caratteristiche. Se nel sistema sono presenti più repository, si dovranno prendere delle precauzioni per essere certi di aver eseguito la migrazione di ogni versione del repository senza danneggiare le informazioni attualmente memorizzate.

Oracle supporta il Simple Network Messaging Protocol (SNMP), e questo supporto consente ai prodotti Oracle di integrarsi agevolmente negli strumenti di monitoraggio per i sistemi e le reti.

Anche se l'impiego di OEM non è necessario, esso fornisce un'interfaccia grafica comune utile per la gestione dei database. Man mano che l'azienda cresce in dimensioni (e nel numero di database e DBA), la coerenza dell'interfaccia del DBA consentirà implementazioni coerenti dei processi di controllo delle modifiche e di controllo della produzione.

### 40.2 Avvio e arresto del database

Per avviare un database, occorre eseguire il comando `startup` dall'interno di SQLPLUS, come mostrato nel listato seguente. Negli esempi di questo capitolo, il nome del database sarà MYDB.

---

```
SQL> connect system/manager as sysdba;
SQL> startup open MYDB;
```

In alternativa, è possibile eseguire innanzi tutto il MOUNT del database e poi aprirlo tramite un comando alter database:

```
SQL> connect system/manager as sysdba;
SQL> startup mount MYDB;
SQL> alter database open;
```

Quando il MOUNT del database è stato eseguito ma il database non è ancora aperto, è possibile gestirne i file. Per esempio, se alcuni file del database sono stati rimossi mentre il database era chiuso, Oracle dovrà sapere dove potrà trovarli prima che il database venga riavviato. Per assegnare le nuove posizioni dei file, è possibile eseguire il MOUNT del database (come mostrato nel listato precedente), e poi usare il comando alter database per rinominare i vecchi file nelle loro nuove posizioni. Dopo aver comunicato a Oracle le nuove posizioni per i file, si potrà aprire il database tramite il comando alter database open.

Per chiudere il database sono disponibili quattro opzioni principali. In una chiusura normale (quella predefinita), Oracle attende che tutti gli utenti si siano disconnessi dal database prima di chiuderlo. In uno shutdown transactional, Oracle attende che le transazioni attive siano state completate prima di chiudere il database. In una chiusura immediata, Oracle annulla tutte le transazioni esistenti non ancora confermate e disconnette qualunque utente ancora connesso. In una chiusura con interruzione, il database si arresta immediatamente e tutte le transazioni non confermate andranno perse.

**NOTA** Durante la fase di avvio o arresto del database non sono consentite nuove connessioni.

Per eseguire una chiusura normale, si utilizza il comando shutdown. Una chiusura immediata, come mostrato nel listato seguente, usa la versione shutdown immediate del comando shutdown. Per terminare il database, si utilizza la versione shutdown abort.

```
SQL> connect system/manager as sysdba;
SQL> shutdown immediate
```

Se si utilizza una chiusura con interruzione, Oracle eseguirà automaticamente le operazioni di recupero durante il successivo avvio del database. Nelle chiusure normali e immediate, il processo di chiusura potrebbe arrestarsi se tra più utenti si verificano dei deadlock. In questi casi, potrebbe essere necessario arrestare la chiusura ed eseguire al suo posto una chiusura con interruzione. Questa soluzione potrebbe rivelarsi utile anche in quelle situazioni in cui il database dev'essere arrestato nel più breve tempo possibile, per esempio a causa di un'imminente interruzione nell'erogazione di corrente elettrica. Durante l'avvio successivo, Oracle eseguirà qualsiasi operazione di recupero si renderà necessaria.

**NOTA** Se si chiude il database per effettuare un backup, è consigliabile utilizzare una chiusura normale oppure una chiusura immediata. Se durante le chiusure si verificano delle complicazioni, si potrebbe ricorrere alla sequenza seguente: chiusura con interruzione, avvio e poi chiusura normale.

Per avviare e chiudere l'istanza è anche possibile usare OEM tramite la scheda General presente nella schermata Instance Manager/Configuration.

### 40.3 Ridimensionamento e gestione delle aree di memoria per il database

Quando si avvia un database, Oracle alloca un'area di memoria (System Global Area, o SGA) condivisa da tutti gli utenti del database. Le due aree più grandi della SGA sono la *database buffer cache* e lo *shared pool*; le loro dimensioni influiranno direttamente sui requisiti di memoria per il database e sulle prestazioni delle operazioni di database. Le loro dimensioni sono controllate da parametri contenuti nel file di inizializzazione del database.

La database buffer cache è un'area nella SGA utilizzata per contenere i blocchi di dati che vengono letti partendo dai segmenti di dati del database, come tabelle, indici e cluster. La dimensione della database buffer cache è determinata dal parametro DB\_CACHE\_SIZE (il cui valore va letto in termini di numero di byte) situato nel file dei parametri di inizializzazione per il database. La dimensione predefinita per i blocchi di database è impostata tramite il parametro DB\_BLOCK\_SIZE che viene specificato nel file dei parametri durante la creazione del database. La gestione della dimensione della database buffer cache è una parte importante della gestione e della messa a punto del database.

Il database ha una dimensione di blocco predefinita, tuttavia si possono stabilire aree della cache per dimensioni di blocchi di database differenti e poi creare delle tablespace per utilizzare proprio queste cache. Per esempio, si può creare un database con una dimensione di blocco di database pari a 4K con alcune tablespace impostate a 8K. La dimensione della cache a 8K verrebbe impostata tramite il parametro DB\_8K\_CACHE\_SIZE. Per generare delle tablespace per utilizzare questa cache, occorre specificare blocksize 8K come parte del comando create tablespace. Se la dimensione di blocco predefinita per il database è pari a 4K, non si dovrebbe impostare un valore per DB\_4K\_CACHE\_SIZE; la dimensione specificata per DB\_CACHE\_SIZE verrebbe utilizzata per la cache a 4K.

Le aree della cache differenti possono essere ridimensionate durante l'esecuzione del database. Le cache dovranno essere aumentate o ridotte in *unità minime*. Per un database con una SGA minore di 128M, la dimensione minima sarà 4M, quindi DB\_8K\_CACHE\_SIZE potrà avere come valori 4M, 8M, 12M e così via. Se si cerca di usare qualsiasi altra impostazione, Oracle la arrotonderà per difetto (alla dimensione minima più vicina). Il listato seguente mostra l'impostazione del parametro DB\_8K\_CACHE\_SIZE.

```
alter system set DB_8K_CACHE_SIZE = 8m;
```

Se si crea una tablespace che utilizza una dimensione di blocco di database non predefinita, occorre essere certi che il parametro corrispondente per la dimensione della cache (per esempio DB\_8K\_CACHE\_SIZE) sia stato aggiornato nel file dei parametri del database. Se si usa un file init.ora, lo si dovrà aggiornare con il nuovo valore. Se invece si utilizza un file dei parametri di sistema, questo verrà aggiornato automaticamente quando si esegue il comando alter system.

**NOTA** *Non è possibile alterare i valori dei parametri SGA\_MAX\_SIZE o JAVA\_POOL\_SIZE mentre il database è aperto.*

Generalmente la buffer cache del blocco di dati occupa all'incirca dall'uno al due per cento della dimensione allocata per il database. Oracle gestirà lo spazio disponibile tramite un algoritmo che mantiene il più a lungo possibile nella memoria i blocchi utilizzati con maggior frequenza. Quando nella cache è necessario dello spazio libero, i nuovi blocchi di dati utilizzeranno lo spazio occupato da blocchi cui si accede raramente oppure quello occupato da un blocco modificato dopo che è stato copiato sul disco.

Se la SGA non è abbastanza grande per contenere i dati più utilizzati, oggetti diversi si contenderanno lo spazio all'interno della buffer cache del blocco di dati. È più probabile che questa situazione si verifichi quando più applicazioni usano il medesimo database e quindi condividono la stessa SGA. In questo caso, le tabelle e gli indici utilizzati con maggiore frequenza da ciascuna applicazione si contendono costantemente lo spazio nella SGA con gli oggetti utilizzati più di recente presi dalle altre applicazioni. Di conseguenza, le richieste di dati dalla buffer cache del blocco di dati comporteranno un rapporto inferiore tra “accessi riusciti” e “accessi non riusciti”. La buffer cache del blocco di dati manca il risultato negli I/O fisici per quanto riguarda le letture dei dati, portando così a un calo delle prestazioni.

Lo shared pool memorizza la cache del dizionario dati (informazioni relative alle strutture di database) e la *library cache* (informazioni su istruzioni che vengono eseguite sul database). Mentre la buffer cache del blocco di dati e la cache del dizionario dati abilitano la condivisione di informazioni strutturali e di informazioni sui dati tra gli utenti nel database, la library cache consente la condivisione di istruzioni SQL utilizzate comunemente.

Lo shared pool contiene il piano di esecuzione e la struttura di analisi per le istruzioni SQL eseguite sul database. Quando un utente qualsiasi esegue per la seconda volta la medesima istruzione SQL, Oracle potrà sfruttare le informazioni di analisi disponibili nello shared pool per velocizzare l'esecuzione dell'istruzione. Come accade per la buffer cache del blocco di dati, anche lo shared pool è gestito tramite un algoritmo LRU. Via via che lo Shared Pool si riempie, i percorsi di esecuzione e le strutture di analisi utilizzati meno recentemente verranno rimossi dalla library cache per far spazio alle nuove voci. Se lo shared pool è troppo piccolo, le istruzioni verranno ricaricate continuamente nella library cache, influendo così sulle prestazioni.

La dimensione (in byte) dello shared pool viene impostata tramite il parametro di inizializzazione SHARED\_POOL\_SIZE. Come per le altre cache, anche la dimensione di Shared Pool può essere alterata mentre il database è aperto.

## Il file dei parametri di inizializzazione

Le caratteristiche dell'istanza di database, come la dimensione della SGA e il numero dei processi in background, vengono specificate durante la fase di avvio. Questi parametri vengono poi memorizzati in un file binario che prende il nome di file dei parametri di sistema. Per generare il file dei parametri di sistema per un database aperto, si usi il comando `create spfile`. Con il comando `create pfile from spfile`, è possibile creare una versione leggibile del file dei parametri di sistema. Si consulti il Capitolo 42 per quanto concerne le sintassi del comando per questi comandi. La versione “pfile” del file dei parametri, conosciuta in precedenza come file `init.ora`, contiene solitamente il nome del database nel nome del file. Per esempio, un database di nome `MYDB` potrebbe avere un file pfile chiamato `initmydb.ora`.

## 40.4 Allocazione e gestione dello spazio per gli oggetti

Per capire in che modo si dovrebbe allocare lo spazio all'interno del database, per prima cosa occorre sapere come viene utilizzato lo spazio all'interno del database. In questo paragrafo, verranno introdotte le funzioni per l'utilizzo dello spazio nel database Oracle.

Quando si crea un database, questo viene diviso in più sezioni logiche che prendono il nome di *tablespace*. La prima tablespace creata è `SYSTEM`. Successivamente, si potranno creare altre tablespace per contenere tipi differenti di dati (come tabelle, indici e segmenti di rollback).

Quando si crea una tablespace, vengono generati anche dei *file di dati* per contenere i suoi dati. Questi file allocano immediatamente lo spazio specificato durante la loro creazione. Ciascun file di dati è in grado di supportare solo una tablespace. È possibile impostare i file di dati in modo che si estendano automaticamente quando esauriscono lo spazio disponibile; si può impostare il loro valore di incremento e la loro dimensione massima. Un database può avere più utenti, ciascuno dei quali ha uno *schema*. Lo schema di ogni utente è una collezione di oggetti di database logici, come tabelle e indici, che fanno riferimento a strutture di dati fisiche memorizzate in tablespace. Gli oggetti presi dallo schema di un utente potrebbero essere memorizzati in più tablespace, e una singola tablespace è in grado di contenere oggetti tratti da più schemi.

Quando si crea un oggetto di database (come una tabella o un indice), questo viene assegnato a una tablespace tramite le impostazioni predefinite dell'utente oppure tramite delle istruzioni specifiche. In questa tablespace si genera un *segmento* per contenere i dati associati a quell'oggetto. Lo spazio allocato per il segmento non verrà mai rilasciato, a meno che il segmento non venga eliminato, ridotto manualmente oppure troncato.

Un segmento è costituito da sezioni chiamate *extent*, ossia gruppi attigui di blocchi Oracle. Quando gli extent esistenti non possono più contenere dati nuovi, il segmento otterrà un altro extent. Il processo di estensione continuerà fino a quando nei file di dati della tablespace non ci sarà più spazio disponibile oppure fino a quando non si raggiungerà un numero interno massimo di extent per segmento. Se un segmento è composto da più extent, non esiste garanzia alcuna che questi extent siano attigui.

È possibile creare le tablespace in modo che vengano gestite a livello locale (impostazione predefinita) oppure gestite a livello di dizionario. In una tablespace gestita a livello di dizionario, le informazioni sulla posizione degli extent verranno memorizzate nel dizionario, mentre in una tabella gestita a livello locale, questi dati verranno memorizzati nelle intestazioni del file di dati. Oracle incoraggia l'uso delle tablespace gestite a livello locale. Per riassumere, i database hanno una o più tablespace, e le tablespace sono costituite da uno o più file di dati. All'interno di queste tablespace, Oracle memorizza segmenti per gli oggetti di database. Ciascun segmento può avere più extent.

La gestione dello spazio utilizzato dalle tablespace, dai file di dati, dai segmenti e dagli oggetti di database è una delle funzioni base del DBA. Lo scopo di questa introduzione generale è aiutare nella pianificazione della loro memorizzazione fisica.

## Implicazioni della storage clause

La quantità di spazio occupata da un segmento è stabilita dai suoi parametri di memorizzazione. Questi parametri sono determinati a loro volta dal database al momento della creazione del segmento; se nei comandi `create table`, `create index`, `create cluster` o `create rollback segment` non sono indicati parametri di memorizzazione specifici, il database utilizzerà i parametri di memorizzazione predefiniti della tablespace in cui dovrà essere memorizzato il segmento.

**NOTA** Agli utenti si possono assegnare tablespace predefinite e assegnare delle quote di spazio all'interno di queste tablespace tramite i comandi `create user`, `alter user` e `grant`. Si consulti il Capitolo 42 per la sintassi dei comandi.

Quando si crea una tabella, un indice oppure un altro segmento, è possibile utilizzare i valori predefiniti di una tablespace gestita a livello locale (opzione consigliata), oppure si può specificare una clausola `storage` nel comando `create`. Inoltre è possibile specificare anche una clausola `tablespace` con la quale si potrà indicare a Oracle di memorizzare i dati in una determinata

tablespace. Per esempio, un comando `create table` in una tablespace gestita a livello di dizionario potrebbe includere le clausole seguenti:

```
tablespace USERS
storage (initial 1M next 1M pctincrease 0
minextents 1 maxextents 200)
```

Per una tablespace `USERS` gestita a livello locale, il comando `create table` dovrebbe includere semplicemente:

```
tablespace USERS
```

I parametri di memorizzazione specificano la dimensione dell'extent iniziale, la dimensione dell'extent successivo, `pctincrease` (il fattore di crescita geometrica di ciascun extent successivo), `maxextents` (il numero massimo di extent) e `minextents` (il numero minimo di extent). Dopo aver creato il segmento, i valori iniziale e `minextents` non potranno essere modificati a meno che non si esegua una riorganizzazione dell'oggetto. Nelle viste `DBA_TABLESPACE` e `USERS_TABLESPACE` sono disponibili i valori predefiniti dei parametri di memorizzazione per ciascuna tablespace.

Quando si crea una tablespace, se ne specificano anche i parametri di memorizzazione predefiniti. Il comando seguente crea una tablespace gestita a livello di dizionario e ne specifica i parametri di memorizzazione predefiniti. Per la sintassi di `create tablespace` si rimanda al Capitolo 42.

```
create tablespace CODES_TABLES
datafile '/u01/oracle/VLDB/codes_tables.dbf' size 10M
extent management dictionary
default storage
  (initial 1M next 1M maxextents 200 pctincrease 0);
```

Quando si crea un segmento, questo acquisirà almeno un extent. L'extent iniziale servirà per memorizzare i dati fino a quando non avrà più spazio libero disponibile (la clausola `pctfree` può essere utilizzata per riservare una certa percentuale di spazio libero all'interno di ciascun blocco nel segmento, in modo che sia disponibile per gli aggiornamenti delle righe già esistenti). Quando al segmento si aggiungono altri dati, esso si estenderà ottenendo un secondo extent con una dimensione specificata dal parametro `next`. Non esiste garanzia alcuna che il secondo extent sia fisicamente adiacente al primo. Nella clausola `storage`, mostrata nel listato precedente, l'extent iniziale ha una dimensione pari a 1MB e l'extent successivo è di 1MB. Dato che `pctincrease` è impostato a 0, ogni extent successivo avrà una dimensione pari a 1MB, fino a un massimo di 200 extent (l'impostazione `maxextents`).

Il parametro `pctincrease` è progettato per ridurre al minimo il numero di extent nelle tabelle in fase di sviluppo. Un valore di questo parametro diverso da zero può essere molto pericoloso, in quanto comporta la crescita geometrica della dimensione di ciascun extent successivo per il fattore `pctincrease` specificato.

**NOTA** *Il parametro `pctincrease` non può essere usato con le tablespace gestite a livello locale.*

Per creare una tabella gestita a livello locale, si specifichi l'opzione `local` per la clausola `extent management` nel comando `create tablespace`. Ecco un esempio del comando `create tablespace` che dichiara una tablespace gestita a livello locale:

```
create tablespace CODES_TABLES
datafile '/u01/oracle/VLDB/codes_tables.dbf'
size 10M
extent management local uniform size 256K;
```

Partendo dal presupposto che in questo esempio la dimensione del blocco per il database in cui è stata creata questa tablespace sia pari a 4KB, la tablespace è stata creata con una gestione degli extent dichiarata come local e con una dimensione uniforme pari a 256KB. Ciascun bit nella bitmap descrive 64 blocchi (256/4). Se la clausola uniform size viene omessa, l'impostazione predefinita sarà autoallocate. La dimensione predefinita per uniform è 1MB.

**NOTA** *Se in un comando create tablespace si specifica local, non si potrà specificare una clausola default storage, minextents o temporary.*

Le tablespace gestite a livello locale sono in grado di assumere alcune delle attività di gestione dello spazio svolte dai DBA nelle tablespace gestite a livello di dizionario. Considerata la loro architettura, sono più difficili da frammentare ed è raro che i loro oggetti presentino dei problemi di spazio.

**NOTA** *In Oracle8i o in Oracle9i Release 1, la tablespace SYSTEM non può essere creata con gestione a livello locale mentre si crea il database, e non potrà nemmeno essere convertita successivamente. A partire da Oracle9i Release 2 (versione 9.2), è possibile creare una tablespace SYSTEM gestita a livello locale.*

Esiste la possibilità di creare un insieme di tablespace gestite a livello locale con un numero ridotto di parametri di memorizzazione e risolvere così la maggior parte delle richieste di spazio nel database. Per esempio, si potrebbero creare tre tablespace DATI, come mostrato di seguito:

```
create tablespace DATI_PICCOLA
datafile '/u01/oracle/VLDB/data_small.dbf'
size 10M
extent management local uniform size 1M;

create tablespace DATI_MEDIA
datafile '/u01/oracle/VLDB/data_medium.dbf'
size 100M
extent management local uniform size 4M;

create tablespace DATA_GRANDE
datafile '/u01/oracle/VLDB/data_large.dbf'
size 1000M
extent management local uniform size 16M;
```

In questo esempio, la tablespace DATI\_PICCOLA crea degli oggetti con dimensioni di extent pari a 1MB; DATI\_MEDIA utilizza dimensioni dell'extent pari a 4MB, mentre DATI\_GRANDE usa dimensioni di extent pari a 16 MB. Se si ha una tabella di piccole dimensioni, sarà sufficiente inserirla nella tablespace DATI\_PICCOLA. Se la sua dimensione aumenta in maniera significativa, la si potrà spostare nelle tablespace DATI\_MEDIA o DATI\_GRANDE.

In ciascuna di queste tablespace di esempio, tutti gli oggetti avranno gli stessi parametri di memorizzazione, semplificando così qualunque operazione di gestione. L'uso di dimensioni coerenti dell'extent migliora anche il riutilizzo dello spazio dopo che un oggetto è stato eliminato o rimosso dalla tablespace.

I prossimi paragrafi descrivono la gestione dello spazio per ciascuno dei principali tipi di oggetti. Si osservi che con le tablespace gestite a livello locale, molti dei problemi legati alle clausole storage non si verificheranno; si farà affidamento sulla tablespace per gestire lo spazio basato sui parametri definiti a livello della tablespace. Anche se ciò potrebbe comportare una certa sovrallocazione di spazio, semplificherà enormemente le operazioni per la gestione dello spazio nel database.

## Segmenti di tabella

I *segmenti di tabella*, chiamati anche *segmenti di dati*, memorizzano le righe di dati associati alle tabelle o ai cluster. Ciascun segmento di dati contiene un blocco di intestazioni che funge da directory di spazio per il segmento.

Dopo aver acquisito un extent, un segmento di dati manterrà questo extent fino a quando non verrà rimosso o eliminato. La cancellazione di alcune righe da una tabella tramite il comando delete non influisce sulla quantità di spazio che è stata allocata per quella tabella. Il numero di extent continuerà ad aumentare fino a quando (1) non si raggiungerà il valore maxextents (se ne è stato impostato almeno uno), (2) non si raggiungerà la quota dell'utente nella tablespace oppure (3) la tablespace esaurirà lo spazio disponibile (nel caso in cui i file di dati non siano in grado di estendersi da soli).

Per ridurre al minimo la quantità di spazio sprecato in un segmento di dati, è possibile regolare il parametro pctfree, che specifica quanto spazio verrà mantenuto libero all'interno di ogni blocco di dati. Questo spazio libero potrà quindi essere impiegato quando le colonne con valore NULL verranno aggiornate per contenere dei valori, oppure quando aggiornamenti di altri valori nella riga comporteranno l'aumento della lunghezza totale della riga. L'impostazione corretta di pctfree è specifica dell'applicazione poiché dipende dalla natura degli aggiornamenti effettuati.

Il comando alter table può servire per alterare moltissimi parametri di memorizzazione di una tabella già esistente. L'opzione move del comando alter table può essere utilizzata per modificare l'assegnamento della tablespace per una tabella.

## Segmenti di indice

Come i segmenti di tabella, anche i *segmenti di indice* contengono lo spazio in essi allocato fino a quando non vengono rimossi; tuttavia, possono essere rimossi anche indirettamente se la tabella o il cluster che indicizzano vengono rimossi. Per ridurre al minimo il conflitto, gli indici dovrebbero essere memorizzati in una tablespace diversa da quella che contiene le loro tabelle associate.

I segmenti di indice contenuti nelle tablespace gestite a livello di dizionario hanno delle clausole storage che ne specificano i valori iniziale (initial), successivo (next), minextents, maxextents e pctincrease e sono meno soggetti a frammentazioni rispetto alle loro tabelle.

L'opzione rebuild del comando alter index può essere utilizzata per alterare le impostazioni storage e tablespace di un indice. Per esempio, se si crea un indice con un extent iniziale troppo grande, è possibile recuperare lo spazio da quell'extent ricreando l'indice e specificando un nuovo valore per initial, come mostrato nell'esempio seguente. Nel listato seguente, l'indice LAVORO\_PK viene ricreato con un extent iniziale di 10MB.

```
alter index LAVORO_PK rebuild
tablespace INDICI
storage (initial 10M next 10M pctincrease 0);
```

Durante il processo di rebuild dell'indice, nel database coesisteranno momentaneamente entrambi gli indici, quello vecchio e quello nuovo. Di conseguenza, è necessario disporre di una quantità di spazio libero sufficiente a memorizzare entrambi gli indici prima di eseguire il comando alter index rebuild.

Durante la costruzione dell'indice, l'ottimizzatore potrà raccogliere statistiche relative al contenuto dell'indice. Si usa la clausola compute statistics dei comandi create index e alter index rebuild per generare le statistiche mentre l'indice viene costruito; l'operazione così combinata sarà più rapida delle due azioni separate di creazione e dei comandi per la raccolta delle statistiche.

## **Segmenti di rollback e undo gestiti a livello di sistema**

I segmenti di rollback possono ridursi in modo dinamico fino a raggiungere una determinata dimensione, oppure possono essere ridotti manualmente a una dimensione di propria scelta. La clausola optimal, che permette ai segmenti di rollback di ridursi fino a raggiungere una dimensione ottimale dopo essersi estesi, aiuta a fornire supporto provvisorio ai sistemi che non sono stati implementati correttamente rispetto al modo in cui sono impiegati. Per gli undo gestiti a livello di sistema (SMU), Oracle alloca e annulla l'allocazione di segmenti undo automaticamente in base alle richieste di elaborazione.

A partire da Oracle9i, la *gestione di undo* automatica potrà essere utilizzata per collocare tutti i dati in un'unica tablespace. Quando si crea una tablespace di undo, Oracle si occupa della memorizzazione, conservazione e utilizzo dello spazio per i dati di rollback tramite gli undo gestiti a livello di sistema (SMU).

Per iniziare a usare gli SMU, innanzi tutto si dovrà creare una tablespace di undo con il comando seguente:

```
create undo tablespace UNDO_TBS datafile 'filespec' size 20m;
```

Successivamente, nel file dei parametri di sistema o nel file init.ora si dovrà impostare il parametro di inizializzazione UNDO\_MANAGEMENT a AUTO. Si specifichi la tablespace di undo tramite il parametro di inizializzazione UNDO\_TABLESPACE. All'interno della tablespace di undo non sarà necessario creare o gestire segmenti di rollback. Si chiuda il database e si utilizzino i file init.ora o spfile.ora aggiornati per avviare il database.

## **Impostazione della conservazione degli undo**

Con il parametro di inizializzazione UNDO\_RETENTION è possibile gestire esplicitamente il lasso di tempo in cui Oracle conserverà i dati di undo nella tablespace di undo. La conservazione degli undo ha due scopi: supporta la conservazione di dati non attivi in uso (IIU) per query a lunga esecuzione e supporta query che mostrano lo stato principale dei dati correnti (query Oracle Flashback).

Per esempio, se si imposta:

```
UNDO_RETENTION=600
```

Oracle farà del suo meglio per conservare nel database tutti i dati di undo su cui è stato eseguito il commit per almeno 600 secondi. Con un'impostazione di questo genere, qualsiasi query che richieda meno di 10 minuti non dovrebbe comportare un errore ORA-1555 ("Snapshot too old"). Durante l'esecuzione del database, è possibile modificare il valore del parametro UNDO\_RETENTION con il comando alter system.

## **Creazione di una tablespace di undo**

Durante il processo di creazione del database è possibile generare una tablespace di undo, come mostrato nel listato seguente:

```
create database UNDODB
undo tablespace UNDO_TBS
...

```

In un database già esistente, è possibile utilizzare il comando `create tablespace` per creare una tablespace di undo:

```
create undo tablespace UNDO_TBS
datafile '/u01/oracle/undodb/undo_tbs_1.dbf'
size 100m;
```

Per la sintassi completa del comando `create tablespace`, si rimanda al Capitolo 42.

Dopo aver creato una tablespace di undo, la si potrà gestire con il comando `alter tablespace`. Per esempio, si potranno aggiungere dei file di dati alla tablespace di undo oppure rinominare i file di dati già esistenti, come si farebbe per qualsiasi altra tablespace. Per bloccare l'uso di una tablespace come tablespace di undo, si usi il comando `alter system` per impostare un nuovo valore per il parametro di inizializzazione `UNDO_TABLESPACE`, oppure si modifichi questo parametro durante una procedura di chiusura/avvio del database.

**NOTA** *All'interno di un database, in un qualsiasi momento non è possibile avere più di una tablespace di undo attiva.*

## **Segmenti temporanei**

I *segmenti temporanei* memorizzano dati temporanei durante le operazioni di ordinamento (come query di grandi dimensioni, creazioni di indici e unioni). Ciascun utente possiede una tablespace temporanea che viene specificata quando si crea l'account tramite `create user`, oppure lo si altera tramite `alter user`. La tablespace temporanea dell'utente dovrà essere una tablespace designata come tablespace temporanea; a partire da Oracle9i, le tablespace permanenti non possono più essere utilizzate per questo scopo.

Quando si crea un database, è possibile specificare una tablespace temporanea predefinita per tutti gli utenti. Per un database già esistente, si potrà modificare la tablespace predefinita con il comando `alter database`, come mostrato nel listato seguente:

```
alter database default temporary tablespace TEMP;
```

Quando si altera l'assegnamento della tablespace temporanea predefinita del database, gli utenti cui era stata assegnata la vecchia tablespace temporanea predefinita verranno reindirizzati alla nuova tablespace temporanea predefinita. Per visualizzare la corrente tablespace temporanea predefinita si esegua la query seguente:

```
select Property_Value
  from DATABASE_PROPERTIES
 where Property_Name = 'DEFAULT_TEMP_TABLESPACE';
```

Con il comando `create temporary tablespace` è possibile specificare una tablespace come "temporanea". Una tablespace temporanea non può essere utilizzata per contenere segmenti per-

manent, ma solo segmenti temporanei creati durante le operazioni. La prima operazione di ordinamento che utilizza la tablespace temporanea alloca un segmento temporaneo nella tablespace temporanea; quando la query è completata, lo spazio usato dal segmento temporaneo non viene rimosso. Al contrario, lo spazio utilizzato dal segmento temporaneo potrà essere usato da altre query, permettendo così all'operazione di ordinamento di evitare i costi di allocazione e rilascio dello spazio per i segmenti temporanei. Se l'applicazione ricorre di frequente ai segmenti temporanei per le operazioni di ordinamento, il processo di ordinamento dovrebbe dare risultati migliori se si utilizza una tablespace temporanea dedicata.

Per dedicare una tablespace già esistente ai segmenti temporanei, si specifichi la clausola `temporary` dei comandi `create tablespace` o `alter tablespace`, come mostrato nel listato seguente:

```
alter tablespace TEMP temporary;
```

**NOTA** *Se in TEMP ci sono dei segmenti permanenti memorizzati (per esempio tabelle o indici), il comando mostrato nel listato seguente fallirà.*

Per permettere alla tablespace TEMP di memorizzare oggetti permanenti (non temporanei), occorre utilizzare la clausola `permanent` dei comandi `create tablespace` o `alter tablespace`, come mostrato nel listato seguente:

```
alter tablespace TEMP permanent;
```

La colonna `Contents` nella vista del dizionario dati `DBA_TABLESPACE` visualizza lo stato della tablespace come “TEMPORARY” o “PERMANENT”.

## Spazio libero

Un *extent libero* in una tablespace è una collezione di blocchi liberi adiacenti nella tablespace. Una tablespace potrebbe contenere più extent di dati ed extent liberi. Quando un segmento viene rimosso, l'allocazione dei suoi extent viene annullata e questi ultimi vengono contrassegnati come liberi. Nelle tablespace gestite a livello di dizionario, non sempre questi extent liberi vengono ricombinati con gli extent liberi adiacenti; le barriere tra questi extent potrebbero rimanere. Il processo di background SMON unisce periodicamente gli extent liberi adiacenti, sempre che il parametro predefinito `pctincrease` per la tablespace sia diverso da zero.

**NOTA** *Il processo di background SMON unisce solamente le tablespace il cui valore `pctincrease` predefinito sia diverso da zero. Un valore `pctincrease` pari a 1 costringerebbe SMON a unire lo spazio libero adiacente in una tablespace, ma unirebbe soltanto otto aree di extent adiacenti alla volta. Ognuno degli otto coalesce si unirebbe a due o più extent per formarne uno più grande. Per una migliore gestione dello spazio, si consiglia di utilizzare le tablespace gestite a livello locale, in modo che non ci si debba mai preoccupare dell'operazione di unione. Se invece sono richieste le tablespace gestite a livello di dizionario, si mantenga un valore `pctincrease` pari a 0 e si uniscano periodicamente gli extent rimossi manualmente.*

Per spingere la tablespace a unire il suo spazio libero, si utilizzi la clausola `coalesce` della clausola `alter tablespace`, come mostrato nel listato seguente:

```
alter tablespace DATI coalesce;
```

Il comando precedente obbligherà gli extent liberi vicini contenuti nella tablespace DATI a unirsi in extent liberi più grossi. Il comando unirà un massimo di otto aree separate, proprio come SMON.

**NOTA** *Il comando alter tablespace non unirà extent liberi che siano separati da extent di dati.*

Le tablespace gestite a livello locale non richiedono lo stesso grado di gestione dello spazio libero. Dato che esiste la possibilità di configurare le tablespace gestite a livello locale in modo che abbiano dimensioni di extent coerenti per tutti i segmenti, gli extent rimossi vengono facilmente riutilizzati.

Quando si alloca un nuovo extent, Oracle non unirà gli extent liberi adiacenti, a meno che non si tratti dell'unica alternativa possibile. Nelle tabelle gestite a livello di dizionario, questo può far sì che un extent di grandi dimensioni collocato verso il fondo della tablespace venga utilizzato, mentre gli extent liberi più piccoli, posizionati all'inizio della tablespace, non vengano quasi mai utilizzati. Nella tablespace, gli extent liberi di piccole dimensioni diventano così dei "riduttori di velocità", in quanto non hanno una dimensione adeguata per essere utili. Man mano che questo modello di uso avanza, la quantità di spazio sprecato nella tablespace aumenta.

In un database ideale, tutti gli oggetti sono creati della loro dimensione appropriata, e tutto lo spazio libero viene sempre memorizzato insieme, un gruppo di risorse pronto all'uso. Se si riesce a evitare l'allocazione dinamica di spazio durante l'utilizzo dell'applicazione, si eliminerebbero un impatto sulle prestazioni e una fonte di potenziali errori dell'applicazione.

## Ridimensionamento degli oggetti di database

La scelta dei parametri adatti per l'allocazione dello spazio per gli oggetti di database è sempre difficile. Gli sviluppatori dovrebbero iniziare a stimare i requisiti di spazio prima ancora di creare i primi oggetti di database. Successivamente, i requisiti di spazio potranno essere ridefiniti in base alle statistiche sull'uso effettivo. I paragrafi successivi analizzano i metodi di valutazione dello spazio per le tabelle, gli indici e i cluster.

### Perché ridimensionare gli oggetti?

Gli oggetti di database dovrebbero essere ridimensionati per tre motivi.

- Per allocare anticipatamente spazio nel database, riducendo così al minimo la quantità di lavoro successivo che sarà necessario per gestire i requisiti di spazio per gli oggetti.
- Per ridurre la quantità di spazio sprecato a causa della sovrallocazione di spazio.
- Per migliorare la probabilità che un extent libero rimosso venga riutilizzato da un altro segmento.

Si potranno raggiungere tutti e tre questi obiettivi seguendo la metodologia di ridimensionamento proposta nei paragrafi successivi. Questa tecnica si basa sui metodi interni di Oracle impiegati per l'allocazione di spazio per gli oggetti di database. Invece di basarsi su calcoli dettagliati, questa metodologia conta su approssimazioni che semplificheranno drasticamente il processo di ridimensionamento insieme alla gestione a lungo termine del database.

### La regola d'oro per i calcoli dello spazio

Mantenere i calcoli dello spazio semplici, generici e coerenti all'interno dei database. Esistono modi molto più produttivi di spendere le proprie ore di lavoro rispetto a eseguire calcoli di spazio estremamente dettagliati che Oracle potrebbe comunque ignorare. Anche se si seguono i

calcoli di ridimensionamento più rigorosi, non si può essere certi del modo in cui Oracle caricherà i dati nella tabella o nell'indice.

Si consideri un indice in una tabella con 100.000 righe. Durante le verifiche, il caricamento dei dati nella tabella secondo un preciso ordine richiederebbe 920 blocchi per l'indice. La tabella e l'indice verrebbero troncati e i dati ricaricati senza un ordine; l'indice ora richiederebbe 1.370 blocchi. Questo aumento di spazio del 49% richiesto dall'indice è stato causato dai processi interni di Oracle per la gestione dell'indice. Se non si è certi di rientrare nel 49% della risposta corretta, perché dedicare così tanto tempo a esercizi di ridimensionamento?

Nel paragrafo successivo, si imparerà a semplificare il processo di valutazione dello spazio, dando così la possibilità di eseguire funzioni DBA molto più utili. Si dovrebbero seguire questi processi sia quando si generano i valori default storage per una tablespace gestita a livello di dizionario, sia quando si creano le dimensioni di extent per le tablespace gestite a livello locale.

### **Le regole fondamentali per i calcoli di spazio**

Quando alloca dello spazio, Oracle segue un insieme di regole interne:

- Oracle alloca solo blocchi interi, non parti di blocchi.
- Oracle alloca gruppi di blocchi, in genere in multipli di cinque blocchi.
- Oracle può allocare gruppi più grandi o più piccoli di blocchi in base allo spazio libero disponibile nella tablespace.

Le prime due regole possono servire come base per stime di spazio abbastanza esatte, ma la terza le annulla entrambe. Anche se si cerca di allocare esattamente 20 blocchi per una tabella, Oracle potrebbe trovare un extent libero di 22 blocchi e utilizzare questo invece di lasciare un extent libero di 2 blocchi.

L'obiettivo dovrebbe essere quello di utilizzare i metodi di allocazione di spazio impiegati da Oracle. Se si usano dimensioni di extent coerenti, si potrà delegare buona parte dell'allocazione di spazio a Oracle.

### **Influsso delle dimensioni di extent sulle prestazioni**

*Riducendo il numero di extent in una tabella, non si ottiene un vantaggio diretto sulle prestazioni.* In alcune situazioni (come in ambienti con query parallele) la possibilità di avere più extent in una tabella può ridurre significativamente i conflitti di I/O e migliorare le prestazioni. A prescindere dal numero di extent nelle tabelle, gli extent dovranno essere ridimensionati in modo corretto.

Oracle legge i dati dalle tabelle in due modi: per RowID (solitamente seguendo immediatamente l'accesso a un indice) e tramite scansioni intere di tabelle. Se i dati sono letti tramite RowID, il numero di extent nella tabella non dev'essere considerato come fattore nelle prestazioni di lettura. Oracle leggerà ogni riga dalla sua posizione fisica (come specificato in RowID) e recupererà i dati.

Se invece i dati sono letti tramite una scansione intera di tabella, allora la dimensione degli extent potrà influire in minima parte sulle prestazioni. Quando si leggono i dati con una scansione intera di tabella, Oracle leggerà più blocchi contemporaneamente. Il numero dei blocchi letti contemporaneamente viene impostato con il parametro di inizializzazione del database DB\_FILE\_MULTIBLOCK\_READ\_COUNT, ed è limitato dalla dimensione del buffer di I/O del sistema operativo. Per esempio, se la dimensione del blocco di database è di 4KB mentre la dimensione del buffer di I/O del sistema operativo è pari a 64 KB, si potranno leggere fino a un massimo di 16 blocchi per lettura durante una scansione intera di tabella. In questo caso, l'impostazione di DB\_FILE\_MULTIBLOCK\_READ\_COUNT a un valore maggiore di 16 non modificherà le prestazioni delle scansioni intere di tabelle.

Le dimensioni di extent dovrebbero sfruttare la capacità di Oracle di eseguire letture di più blocchi durante le scansioni intere di tabelle. Quindi, se il buffer di I/O del sistema operativo è di 64 KB, le dimensioni di extent dovrebbero essere un multiplo di 64KB.

Si consideri una tabella con dieci extent, ciascuno dei quali ha una dimensione pari a 64 KB. Per questo esempio, la dimensione del buffer di I/O del sistema operativo sarà di 64KB. Per eseguire una scansione intera di tabella, Oracle dovrà eseguire dieci letture (dato che 64KB è la dimensione del buffer di I/O del sistema operativo). Se i dati sono compressi in un unico extent di 640KB, Oracle dovrà comunque eseguire dieci letture per eseguire la scansione della tabella. La compressione degli extent non comporta alcun miglioramento delle prestazioni.

Se la dimensione di extent della tabella non è un multiplo della dimensione del buffer di I/O, il numero di letture richieste potrebbe aumentare. Per la stessa tabella di 640KB si potrebbero creare otto extent di 80KB ciascuno. Per leggere il primo extent, Oracle eseguirà due letture: una per i primi 64KB dell'extent e una seconda lettura per i rimanenti 16KB dell'extent (le letture non possono superare gli extent). Per leggere l'intera tabella, Oracle dovrà effettuare due letture per extent, o 16 letture. La riduzione del numero di extent da 10 a 8 ha comportato un aumento delle letture pari al 60%.

Per evitare che le dimensioni di extent provochino un calo delle prestazioni, si dovrà scegliere tra una delle strategie seguenti.

1. Creare extent che siano molto più grandi della dimensione di I/O. Se gli extent sono molto grandi, le letture aggiuntive necessarie saranno davvero poche anche se la dimensione di extent non è un multiplo della dimensione del buffer di I/O.
2. Impostare DB\_FILE\_MULTIBLOCK\_READ\_COUNT in modo da sfruttare pienamente la dimensione del buffer di I/O a favore del sistema operativo. Un'impostazione troppo alta di questo valore potrebbe indurre l'ottimizzatore a pensare che le scansioni intere di tabelle siano più efficienti di quello che in realtà sono, provocando così dei cambiamenti ai piani di esecuzione già esistenti.
3. Se si devono creare extent di piccole dimensioni, si scelgano dimensioni di extent che siano multipli della dimensione del buffer di I/O del sistema operativo.

Se la dimensione del buffer di I/O per il sistema operativo è di 64KB, si potrà scegliere tra dimensioni di extent come 64KB, 128KB, 256KB, 512KB, 1MB e così via. Nel prossimo paragrafo, si imparerà a ridurre ulteriormente la gamma di dimensioni di extent tra cui scegliere.

Per massimizzare il riutilizzo degli extent liberi rimossi nelle tablespace gestite a livello di dizionario, si utilizzi un insieme di dimensioni di extent che soddisfi il criterio seguente: *ogni dimensione di extent conterrà un multiplo intero per ogni dimensione di extent più piccola*. L'implementazione più semplice di questa regola consiste nel creare dimensioni di extent che aumentano esattamente del doppio: 1MB, 2MB, 4MB, 16MB, 32MB. Per ridurre il numero delle dimensioni di extent da gestire, si possono quadruplicare i valori invece di raddoppiarli: 1MB, 4MB, 16MB e così via. L'uso di questi valori per le dimensioni di extent eliminerà i problemi di I/O e migliorerà le possibilità di riutilizzare gli extent rimossi.

## Ridimensionamento degli oggetti

Per gestire in modo efficiente lo spazio, è sufficiente selezionare un insieme di valori di spazio che soddisfano i criteri descritti nei paragrafi precedenti. Dopo aver finalizzato le allocazioni di spazio, le si separi con una tablespace. Per esempio:

```
create tablespace DATA_1M
datafile '/u01/oracle/VLDB/dati_1m.dbf'
size 100M
extent management local uniform size 1M;
```

```

create tablespace DATI_4M
datafile '/u02/oracle/VLDB/dati_4m.dbf'
size 400M
extent management local uniform size 4M;

create tablespace DATI_16M
datafile '/u03/oracle/VLDB/dati_16m.dbf'
size 16000M
extent management local uniform size 16M;

```

In questo esempio, vengono create tre tablespace DATI separate, con dimensioni di extent pari a 1MB, 4MB e 16MB. Se si deve generare una tabella con una dimensione di 3MB, la si potrà creare utilizzando tre extent da 1MB in DATI\_1M oppure con un extent da 4MB in DATI\_4M. Una tabella che raggiungerà i 10MB potrà essere inserita in DATI\_16M.

Man mano che le dimensioni delle tabelle aumentano, le clausole di memorizzazione predefinite cresceranno in modo coerente, seguendo le regole di spazio e gli standard per le dimensioni di extent. Sarà la volta di DATI\_64M, seguito da DATI\_256M e DATI\_1G. Si utilizzi le stesse dimensioni di extent tra i database per semplificare la gestione dello spazio dell'intero ambiente di database.

Mentre le dimensioni di extent crescono, la distribuzione delle medesime nelle tablespace comporterà una separazione dei tipi di tabelle: le tabelle statiche di piccole dimensioni verranno isolate nelle tablespace con dimensioni di extent piccole, mentre le tabelle per l'elaborazione di grandi transazioni (o le loro partizioni) verranno segregate nelle tabelle con dimensioni di extent elevate, semplificando le future attività di gestione e messa a punto.

Nei paragrafi seguenti verranno descritte delle linee guide utili per le stime relative all'uso dello spazio per i vari oggetti. Dato che le dimensioni di destinazione (1MB, 4MB, 16MB e così via) non sono molto vicine, le stime seguenti non includeranno calcoli estremamente dettagliati.

**NOTA** *A partire da Oracle9i, la dimensione di blocco del database può cambiare a livello di tablespace. Si controlli che le dimensioni di extent scelte valgano per la dimensione di blocco più grande utilizzata nel database. La limitazione dell'uso di dimensioni di blocco non standard nel database semplificherà la gestione tra database e le procedure di ridimensionamento.*

## 40.5 Creazione e gestione dei segmenti di rollback

Come sottolineato nel paragrafo sui segmenti di rollback e undo gestiti a livello di sistema, inserito nella parte di questo capitolo dedicata alla gestione dello spazio, i segmenti di rollback si espandono e contraggono dinamicamente per supportare le transazioni. Quando si crea un database, Oracle genera automaticamente un segmento di rollback SYSTEM che supporta le transazioni all'interno del dizionario dati. È necessario creare un secondo segmento di rollback e lo si dovrà mettere in linea prima di utilizzare qualsiasi tablespace non SYSTEM. Si vedano gli script del comando create database creati da OUI per esempi del comando create rollback segment.

In linea di massima, si dovrebbe fare attenzione alle procedure seguenti che coinvolgono segmenti di rollback:

- Come metterli in linea e fuori linea.
- Come stabilire la loro dimensione massima.

- Come assegnare le transazioni a segmenti di rollback specifici.

Questi tre passaggi permetteranno agli sviluppatori di supportare in modo corretto le dimensioni di transazione richieste per le parti batch e in linea delle applicazioni.

## Attivazione dei segmenti di rollback

L'*attivazione* di un segmento di rollback lo rende disponibile agli utenti del database. Un segmento di rollback potrebbe venire disattivato senza essere rimosso. Un segmento di rollback disattivato manterrà lo spazio che gli è stato allocato, e potrà essere riattivato successivamente. Gli esempi seguenti forniscono il gruppo completo dei comandi per l'attivazione dei segmenti di rollback.

Con il comando `alter rollback segment` è possibile disattivare un segmento di rollback attivo.

```
alter rollback segment NOME_SEGMENTO offline;
```

Per rimuovere un segmento di rollback, si utilizzi il comando `drop rollback segment`.

```
drop rollback segment NOME_SEGMENTO;
```

Per creare un segmento di rollback, si utilizzi il comando `create rollback segment`, come mostrato nel listato seguente:

```
create rollback segment NOME_SEGMENTO
tablespace RBS;
```

Si osservi che il comando di esempio `create rollback segment` crea un segmento di rollback privato (dato che la parola chiave `public` non è stata utilizzata) e che lo crea in una tablespace non `SYSTEM`, di nome `RBS`. Dato che non sono stati specificati parametri di memorizzazione, il segmento di rollback si servirà dei parametri di memorizzazione predefiniti impostati per quella tablespace (più avanti nel capitolo si analizzeranno i dettagli della gestione dello spazio per i segmenti di rollback).

Anche se il segmento di rollback è già stato creato, il database non lo utilizza ancora. Per attivare il nuovo segmento di rollback, occorre metterlo in linea con il comando seguente:

```
alter rollback segment NOME_SEGMENTO online;
```

Dopo aver creato un segmento di rollback, lo si dovrebbe elencare nel file dei parametri di inizializzazione del database. Nel listato seguente è mostrata una voce di inizializzazione di esempio per i segmenti di rollback:

```
rollback_segments = (r0,r1,r2)
```

**NOTA** Il segmento di rollback `SYSTEM` non dovrebbe essere mai elencato nel file dei parametri di inizializzazione. Infatti, il segmento di rollback `SYSTEM` non potrà mai essere rimosso; viene sempre acquisito insieme agli altri segmenti di rollback che l'istanza può acquisire.

Nell'esempio precedente, i segmenti di rollback chiamati `r0`, `r1` e `r2` sono in linea. Se si mette fuori linea un segmento di rollback, si dovrà rimuovere la sua voce dal file dei parametri. Quando si mette fuori linea un segmento di rollback, questo rimarrà in linea fino a quando le sue transazioni attive non saranno state completate.

## Come stabilire la dimensione massima di un segmento di rollback

Un segmento di rollback è un segmento contenuto nel database, quindi è possibile eseguire una query dei suoi valori di memorizzazione dalla vista del dizionario dati DBA\_SEGMENTS. Quando si usano i segmenti di rollback, è importante sapere di quanto potrà crescere il segmento di rollback. Dato che una transazione non può superare i segmenti di rollback, questi dovranno essere grandi a sufficienza per supportare la transazione più grande che verrà effettuata. Come descritto nel prossimo paragrafo, è possibile indurre Oracle a usare un segmento di rollback specifico per le transazioni.

Con una query su DBA\_SEGMENTS è possibile stabilire la dimensione massima di un segmento di rollback:

```
select Segment_Name,
       Tablespace_Name,
       Bytes AS Current_Size,
       Initial_Extent + Next_Extent*(Max_Extents-1)
              AS Max_Size
  from DBA_SEGMENTS
 where Segment_Type = 'ROLLBACK';
```

Il risultato di questa query dà la dimensione massima per i segmenti di rollback, basandosi sulle definizioni dei segmenti. Successivamente si dovrebbe confrontare questo valore con lo spazio libero contenuto nei file di dati per la tablespace del segmento di rollback. Se autoextend è abilitato per i file di dati, la dimensione dei file di dati del segmento di rollback potrebbe essere limitata dalla quantità di spazio disponibile su disco oppure dall'impostazione maxsize per i file di dati.

Quando possibile, si limiti la dimensione delle transazioni batch. Più piccola è una transazione, più facile sarà la sua gestione all'interno del database. Se si deve avere una transazione di grandi dimensioni, la si dovrebbe isolare in una tablespace specifica, come descritto nel prossimo paragrafo. Se non è possibile assegnare la transazione a un segmento di rollback, tutti i segmenti di rollback non SYSTEM dovranno essere abbastanza grandi per supportare questa transazione.

Se un segmento di rollback deve estendersi di frequente oltre la sua impostazione optimal, questa gestione dinamica dello spazio potrebbe influire sulle prestazioni delle transazioni. Si imposti il parametro di memorizzazione optimal per i segmenti di rollback in modo che rifletta l'uso medio dello spazio all'interno del segmento.

## Monitoraggio di una tablespace di undo

Per gli undo gestiti a livello di sistema, V\$UNDOSTAT è la vista principale per raccogliere le statistiche sull'utilizzo delle tablespace di undo. La vista V\$UNDOSTAT fornisce un istogramma sull'uso dei dati mantenuti. A intervalli di dieci minuti, Oracle registra il numero totale dei blocchi di undo consumati nella colonna UndoBlks di V\$UNDOSTAT, e il numero di transazioni supportate nella colonna TxnCount. I dati tratti da V\$UNDOSTAT possono servire per stabilire se la tablespace di undo ha una dimensione adatta per il volume di transazioni dell'applicazione. Oracle mantiene 144 righe in V\$UNDOSTAT, riflettendo così le statistiche del giorno precedente.

Tra le altre colonne di un certo interesse in V\$UNDOSTAT c'è MaxQuerylen, che registra la durata (in secondi) della query più lunga che è stata eseguita; la conservazione degli undo de-

v'essere maggiore di questo valore. Se la colonna NoSpaceErrCnt di V\$UNDOSTAT ha un valore diverso da zero, è consigliabile aggiungere dello spazio alla tablespace di undo per evitare futuri errori legati proprio allo spazio.

## Come assegnare le transazioni a segmenti di rollback specifici

Con il comando set transaction si può specificare quale segmento di rollback dovrebbe utilizzare una transazione. Il comando set transaction dovrebbe essere eseguito prima delle transazioni di grandi dimensioni per essere certi che queste utilizzino i segmenti di rollback creati appositamente per loro.

Le impostazioni specificate tramite il comando set transaction verranno usate solamente per la transazione corrente. L'esempio seguente mostra una serie di transazioni. Alla prima transazione viene data l'istruzione di utilizzare il segmento di rollback ROLL\_BATCH. La seconda transazione (che segue il secondo commit) verrà assegnata a caso a un segmento di rollback di produzione.

```
commit;
```

```
set transaction use rollback segment ROLL_BATCH;
insert into NOME_TABELLA
select * from TABELLA_CARICAMENTO_DATI;
```

```
commit;
```

REM\* Il comando commit annulla l'assegnamento del segmento di rollback.

REM\* I comandi commit impliciti, come quelli causati dai comandi DDL,

REM\* annulleranno anche la designazione del segmento di rollback.

```
insert into NOME_TABELLA select * from QUALCHE_ALTRA_TABELLA;
```

## 40.6 Esecuzione dei backup

Come per gli altri argomenti introdotti in questo capitolo, anche questo paragrafo si limiterà a offrire un'introduzione. Il backup e il recupero sono argomenti complessi, quindi tutti i metodi di backup e recupero dovrebbero essere verificati e provati accuratamente prima di essere implementati in un ambiente produttivo. Lo scopo di questo paragrafo è fornire agli sviluppatori una conoscenza delle capacità di Oracle insieme all'impatto che le decisioni di backup hanno sugli sforzi e la disponibilità del recupero. Prima di utilizzare i metodi di backup e recupero in un ambiente produttivo, è consigliabile consultare libri dedicati esclusivamente a questo argomento, non trascurando la documentazione di Oracle.

I metodi di backup messi a disposizione da Oracle possono essere classificati nel modo seguente:

- Backup logici che utilizzano Export (e la sua utility complementare, Import).
- Backup di file system fisici: backup in linea e backup fuori linea.
- Backup di file system fisici incrementali effettuati tramite Recovery Manager (RMAN).

I paragrafi successivi descrivono ciascuno di questi metodi e le rispettive capacità.

## Export e Import

L'utility Export di Oracle esegue una query sul database, compreso il dizionario dati, e scrive l'output su un file binario che prende il nome di *file di dump di esportazione*. È possibile esportare l'intero database, utenti specifici oppure tabelle specifiche. Durante le operazioni di esportazione, si potrà decidere se esportare o meno le informazioni del dizionario dati associate alle tabelle, come concessioni, indici e vincoli a esse associati. Il file scritto da Export conterrà i comandi necessari per ricreare completamente tutti gli oggetti e i dati scelti.

A partire da Oracle9i è possibile effettuare esportazioni a livello di tablespace per esportare tutti gli oggetti contenuti in una tablespace. Qualunque indice specificato sulle tabelle esportate verrà esportato. Le esportazioni a livello di tablespace utilizzano la clausola tablespaces dell'utility Export, come verrà descritto più avanti in questo capitolo.

Dopo essere stati esportati, i dati possono essere importati tramite l'utility Import di Oracle. L'utility Import legge il file di dump di esportazione binario creato da Export ed esegue i comandi che trova in questo file. Per esempio, questi comandi potrebbero includere `create table` seguito da un comando `insert` per caricare dei dati nella tabella.

La consuetudine di generare un file di dump di esportazione non implica che i dati esportati debbano essere importati obbligatoriamente nel medesimo database, o nello stesso schema. Il file di dump di esportazione potrebbe servire per creare un gruppo di duplicati degli oggetti esportati in uno schema differente o in un database separato.

Si possono importare tutti i dati esportati oppure solo una parte di essi. Se si importa l'intero file di dump di esportazione da un'esportazione Full, allora tutti gli oggetti di database, tra cui le tablespace, i file di dati e gli utenti verranno creati durante l'importazione. Tuttavia, spesso è utile creare in anticipo tablespace e utenti per specificare la distribuzione fisica degli oggetti nel database.

Se invece si importerà solo una parte dei dati dal file di dump di esportazione, le tablespace, i file di dati e gli utenti che possiederanno e memorizzeranno questi dati dovranno essere configurati prima dell'importazione.

I dati esportati potrebbero essere importati in un database creato in una versione successiva del software Oracle. L'operazione inversa, l'importazione da un file di esportazione creato da una release più recente, viene supportata, anche se potrebbero rendersi necessarie ulteriori operazioni per supportare la versione più datata delle viste utilizzate da Export. I file README.doc che accompagnano le release di Oracle descrivono in dettaglio i requisiti per l'utilizzo di Export e Import tra le varie versioni specifiche.

## Export

L'utility Export ha quattro livelli di funzionalità, definiti dalle modalità Full, Tablespace, User e Table. Si possono esportare delle partizioni tramite una versione modificata delle esportazioni di modalità Table.

Nella modalità Full, viene esportato l'intero database. L'intero dizionario dati viene letto e il DDL necessario per ricreare l'intero database viene scritto nel file di dump di esportazione. Questo file comprende comandi di creazione per tutte le tablespace, tutti gli utenti, tutti gli oggetti, i dati e i privilegi contenuti nei loro schemi.

Nella modalità Tablespace, verranno esportati tutti gli oggetti contenuti nelle tablespace specificate, compresa la definizione degli indici relativi agli oggetti contenuti, anche se questi si trovano in un'altra tablespace.

Nella modalità User, vengono esportati gli oggetti di un utente, così come i dati in essi contenuti. Inoltre, vengono esportati anche tutti gli indici e le concessioni creati dall'utente sui

propri oggetti. Le concessioni e gli indici creati da utenti che non sono i proprietari non vengono esportati.

Nella modalità Table viene esportata una tabella specificata. La struttura, gli indici e le concessioni della tabella vengono esportati con o senza i suoi dati. La modalità Table può esportare anche l'intero gruppo di tabelle possedute da un utente (specificando il possessore dello schema ma non i nomi delle tabelle). Inoltre, si possono specificare partizioni di una tabella da esportare.

Export può essere eseguita in modo interattivo tramite Oracle Enterprise Manager oppure con i file dei comandi. Le opzioni runtime che possono essere specificate per Export sono elencate nella Tabella 40.1, insieme ai loro valori predefiniti in Oracle9i.

Alcuni parametri sono in reciproco conflitto oppure potrebbero portare a istruzioni incoerenti per Export. Per esempio, l'impostazione full=Y e owner=HR fallirebbe, perché il parametro full chiama un'esportazione Full, mentre il parametro owner specifica un'esportazione User.

**NOTA** *I valori predefiniti e i parametri disponibili possono cambiare con ogni release di Oracle.*

Con il comando seguente è possibile visualizzare i parametri di Export in linea:

```
exp help=Y
```

L'opzione compress=Y altera il parametro initial della clausola storage per quei segmenti che hanno più extent. Durante una successiva importazione, lo spazio totale allocato per quel segmento verrà compresso in un unico extent. Riguardo a questa funzionalità ci sono due considerazioni importanti da fare.

- Innanzitutto, è lo spazio *allocato* e non quello *utilizzato* che viene compresso. Una tabella vuota per la quale sono stati allocati 300MB in tre extent da 100MB verrà compressa in un unico extent vuoto di 300MB. Non verrà reclamato dello spazio.
- In secondo luogo, se la tablespace ha più file di dati, un segmento potrebbe allocare uno spazio maggiore rispetto alla dimensione del file di dati più grande. In questo caso, l'uso di compress=Y modificherebbe la clausola storage in modo che abbia una dimensione di extent iniziale maggiore di qualsiasi dimensione del file di dati. Dato che un singolo extent non può estendersi su più di un file di dati, la creazione dell'oggetto fallirà durante l'importazione.

Nell'esempio seguente, viene utilizzata l'opzione compress=Y mentre i proprietari HR e THUMPER vengono esportati:

```
exp system/manager file=expdat.dmp compress=Y owner=(HR,THUMPER)
```

**NOTA** *Con le tablespace gestite a livello locale si possono evitare i problemi di frammentazione dello spazio.*

### Esportazioni coerenti

Durante la scrittura dei dati di database nel file di dump di esportazione, Export legge una tabella per volta. Quindi, anche se Export iniziasse a un orario ben specifico, ciascuna tabella verrebbe letta a orari diversi. Ciò che verrà esportato sono i dati così come si trovano in ogni tabella nel momento in cui Export inizia a leggere *quella tabella*. Considerata la correlazione che lega la maggior parte delle tabelle ad altre tabelle, si potrebbero verificare esportazioni di dati incoerenti se gli utenti modificano i dati durante l'esportazione. Il file di dump di esportazione potrebbe contenere dei dati incoerenti, come record di chiave esterna presi da una tabella senza record di chiave primaria corrispondenti presi da una tabella correlata.

**Tabella 40.1** Le opzioni runtime di Export.

PAROLA RISERVATA	DESCRIZIONE
userid	Nome utente/password dell'account che esegue l'esportazione. Dev'essere il primo parametro sulla linea di comando.
buffer	Dimensioni del buffer utilizzato per prelevare i dati. Il valore predefinito dipende dal sistema; di solito a questo parametro viene assegnato un valore alto (maggiore di 64000).
file	Nome del file di dump di esportazione: il nome predefinita è expdat.dmp.
compress	Un flag Y/N per indicare se l'esportazione deve comprimere i segmenti frammentati in un unico extent. Questo ha un effetto sulle clausole di memorizzazione storage che vengono salvate nel file di esportazione per questi oggetti. Il valore predefinito è Y. Avere i dati in un solo extent di grandi dimensioni non sempre è la soluzione ideale. Per le tabelle di grandi dimensioni si dovrebbe utilizzare compress =N.
grants	Un flag Y/N per indicare se devono essere esportate anche le concessioni (grant) sugli oggetti del database). Il valore predefinito è Y.
indexes	Un flag Y/N per indicare se devono essere esportati anche gli indici delle tabelle. Il valore predefinito è Y.
direct	Un flag Y/N per indicare se dev'essere effettuata un'esportazione diretta. Un'esportazione diretta aggira la buffer cache durante l'esportazione, producendo un significativo aumento delle prestazioni del processo di esportazione. Il valore predefinito è N.
log	Nome del file nel quale verrà scritto il log dell'esportazione.
rows	Un flag Y/N per indicare se le righe devono essere esportate. Se si usa N, nel file di esportazione viene creato solo il codice DDL relativo agli oggetti del database. Il valore predefinito è Y.
consistent	Un flag Y/N per indicare se dev'essere mantenuta una versione coerente di tutti gli oggetti esportati. Questo parametro è necessario quando delle tabelle correlate fra loro potrebbero venire modificate dagli utenti durante il processo di esportazione.
full	Se si usa Y viene effettuata un'esportazione completa del database.
owner	Un elenco di account del database da esportare; dopodiché possono essere effettuate esportazioni in modalità utente di questi account.
tables	Un elenco delle tabelle da esportare; quindi possono essere effettuate esportazioni in modalità a tabella di queste tabelle. A partire da Oracle9i questo parametro accetta l'uso dei caratteri jolly % e _ per il pattern matching.
recordlength	La lunghezza, in byte, dei record dei file di dump di Export. Di solito viene impiegato il valore standard, tranne quando si devono trasferire i file da un sistema operativo a un altro.
triggers	Un flag Y/N per indicare se i trigger devono essere esportati. Il valore predefinito è Y.
statistics	Un parametro per indicare se sul file di dump di Export devono essere scritti i comandi analyze relativi agli oggetti esportati. I valori validi sono "COMPUTE", "ESTIMATE," (l'impostazione standard) e N. Nelle precedenti versioni di Oracle questo parametro era chiamato "analyze".
parfile	Il nome di un file di parametri da passare a Export. Questo file può contenere tutti i parametri elencati in questa tabella.
constraints	Un flag Y/N per indicare se devono essere esportate le restrizioni sulle tabelle. Il valore predefinito è Y.
feedback	Il numero di righe dopo cui visualizzare la progressione dell'esportazione delle tabelle. Il valore standard è 0; in questo modo non viene mostrata alcuna informazione finché una tabella non sia stata completamente esportata.

(segue)

**Tabella 40.1** Le opzioni runtime di Export. (*continua*)

PAROLA RISERVATA	DESCRIZIONE
filesize	La massima dimensione del file di dump di esportazione. Se sono elencati più file nella voce "file", l'esportazione sarà indirizzata a tali file in relazione alla dimensione di filesize.
flashback_scn	Specifica l'SCN che Export utilizzerà per attivare il flashback. L'esportazione avverrà con i dati coerenti a partire da questo SCN.
flashback_time	Tempo usato per prendere l'SCN più vicino all'ora specificata. L'esportazione avverrà con i dati coerenti a partire da questo SCN.
query	Una clausola where che verrà applicata a ogni tabella durante l'esportazione.
resumable	Un flag Y/N per indicare se la sessione può essere recuperata in seguito a degli errori. Il valore predefinito è N.
resumable_name	Il valore specificato viene inserito nella vista DBA_RESUMABLE per aiutare a identificare l'istruzione recuperabile.
resumable_timeout	Il tempo di attesa per le istruzioni recuperabili.
tts_full_check	Effettua un controllo delle dipendenze completo o parziale per individuare le tablespace trasportabili.
volsize	Numero di byte da scrivere per ciascun volume del nastro.
tablespaces	In Oracle9i le tablespace le cui tabelle dovrebbero essere esportate, comprese tutte le tabelle che non hanno una partizione situata nelle tablespace specificate.
transpor_tablespace	Impostata a Y se si usa l'opzione pluggable tablespace disponibile a partire da Oracle8i. Da utilizzare insieme alla parola riservata tablespaces. Il valore predefinito è N.
template	Nome del modello utilizzato per chiamare l'esportazione in modalità iAS.

Per evitare incoerenze nelle esportazioni esistono due possibilità:

- Innanzitutto, si dovrebbero programmare le esportazioni in modo che avvengano quando nessuno sta apportando modifiche alle tabelle. Se fattibile, si può utilizzare l'opzione startup restrict per assicurarsi che solo i DBA siano connessi mentre avviene l'esportazione.
- In secondo luogo, si può usare il parametro consistent. Questo parametro è disponibile solo per esportazioni complete. Durante un'esportazione coerente, Oracle cercherà di creare una versione a lettura coerente dei dati esportati a partire dal momento in cui è iniziata l'esportazione. Se Oracle non è in grado di ricreare la versione a lettura coerente delle tabelle, compariranno errori "snapshot too old". Per essere certi di non incontrare errori di questo genere, si dovranno creare segmenti di rollback molto grandi oppure una tablespace di undo di grandi dimensioni, perché Oracle sovrascriverà le voci dei vecchi segmenti di rollback ignorando l'impostazione consistent=Y.
- In terzo luogo, si potrebbe rendere inattivo il database, operazione che limita le connessioni ai soli utenti con privilegi SYSDBA. Mentre il database è quiescente, tutte le transazioni e le query in corso sembra che siano in attesa, e non sono ammesse connessioni di nuovi utenti.

Ogni volta che è possibile, si garantisca la coerenza dei dati esportati eseguendo le esportazioni mentre il database non è utilizzato, oppure quando il suo MOUNT è in modalità limitata. Se ciò non è possibile, si limiti l'utilizzo del database durante la fase di esportazione e si esegua un'esportazione consistent=Y.

Un'altra possibilità è la creazione di due file dei parametri (parfile). Un file conterrà i nomi della maggior parte delle tabelle presenti nel database oppure nello schema; si potrà ricorrere a questo file per la prima esportazione con l'impostazione predefinita `consistent=N`. Il secondo file, contenente i nomi delle tabelle per cui si dovrà mantenere la coerenza, serve per eseguire un'esportazione con l'opzione `consistent=Y`. Con questo approccio, la dimensione dei segmenti di rollback non dovrà essere così grande, e le prestazioni del sistema non soffriranno troppo.

### **Esportazione delle tablespace**

Una migliore distribuzione dei proprietari di schema dell'applicazione e dei loro oggetti nelle tablespace garantisce un'esportazione più semplice dei medesimi. Per esempio, se si intende esportare tutti gli oggetti nello schema AP\_DATI, si potrà eseguire un'esportazione User se un solo utente possiede gli oggetti in quella tablespace. Se più utenti possiedono oggetti nella tablespace DATI, si potranno esportare tutti gli oggetti nella tablespace con un comando (iniziano con Oracle9i) indipendentemente dallo schema che possiede l'oggetto:

```
exp demo/demo tablespaces=DATI
```

La query seguente associa gli utenti alle tablespace per stabilire la distribuzione delle loro tabelle e dei loro indici:

```
break on Owner on Tablespace_Name
column Objects format A20
select
    Owner,
    Tablespace_Name,
    COUNT(*)||' tables' Objects
  from DBA_TABLES
 where Owner <> 'SYS'
group by
    Owner,
    Tablespace_Name
union
select
    Owner,
    Tablespace_Name,
    COUNT(*)||' indexes' Objects
  from DBA_INDEXES
 where Owner = 'SYS'
group by
    Owner,
    Tablespace_Name;
```

Ecco un output di esempio tratto da questa query:

OWNER	TABLESPACE_NAME	OBJECTS
FLOWER	USERS	3 tables 2 indexes
HR	HR_TABLES	27 tables
	HR_INDEXES	35 indexes
THUMPER	USERS	5 tables

L'output di esempio riportato nel listato precedente mostra che l'account utente FLOWER possiede tabelle e indici nella tablespace USERS e che anche THUMPER possiede diverse ta-

belle in quella tablespace. L'utente HR possiede oggetti in entrambe le tablespace HR\_TABLES e HR\_INDEXES.

Prima di stabilire le combinazioni corrette degli utenti da esportare per una tablespace, si dovrebbe rivedere l'associazione inversa, tablespace a utenti, con la query seguente:

```
break on Tablespace_Name on Owner
column Objects format A20
select
    Tablespace_Name,
    Owner,
    COUNT(*)||' tables' Objects
  from DBA_TABLES
 where Owner <> 'SYS'
group by
    Tablespace_Name,
    Owner
union
select
    Tablespace_Name,
    Owner,
    COUNT(*)||' indexes' Objects
  from DBA_INDEXES
 where Owner <> 'SYS'
group by
    Tablespace_Name,
    Owner;
```

Nel listato seguente è mostrato dell'output di esempio preso da questa query:

TABLESPACE_NAME	OWNER	OBJECTS
HR_INDEXES	HR	35 indexes
HR_TABLES	HR	27 tables
USERS	FLOWER	3 tables
		2 indexes
	THUMPER	5 tables

L'output di esempio riportato nel listato precedente mostra che la tablespace HR\_TABLES contiene oggetti di un solo utente (l'account HR). La tablespace HR\_INDEXES viene isolata in modo analogo. Dall'altra parte, la tablespace USERS contiene tabelle e indici presi da diversi account.

I risultati delle query precedenti illustrano l'importanza di una corretta distribuzione degli oggetti degli utenti tra le varie tablespace. Dato che la tablespace HR\_TABLES contiene solamente tabelle possedute dall'account HR (dalla seconda query), l'esportazione delle tabelle di HR esporterà tutti gli oggetti contenuti nella tablespace HR\_TABLES. Come si è visto già dalla prima query, HR non ha tabelle in nessun'altra parte del database. Dato che le tabelle di HR sono isolate nella tablespace HR\_TABLES, e siccome questa tablespace è utilizzata solamente dall'account HR, un'esportazione User per HR esporterà tutte le tabelle in HR\_TABLES. Dato che gli indici di queste tabelle sono memorizzati in HR\_INDEXES, questa tablespace potrà essere ricreata nello stesso istante dallo stesso file di dump di esportazione.

A partire da Oracle9i, si potrà utilizzare il parametro tablespaces per esportare tutte le tabelle che si trovano in una tablespace specifica. Se una tabella ha una partizione nella tablespace specificata, verrà esportata l'intera tabella. Se si imposta indexes=Y, gli indici associati della tabella verranno esportati indipendentemente dalla loro posizione nella tablespace.

## Esportazione di partizioni

Quando si eseguono esportazioni Table, è possibile fare riferimento alle partizioni e alle partizioni secondarie all'interno delle tabelle. Per esempio, se la tabella VENDITE nello schema THUMPER è suddivisa in PARTE1, PARTE2 e PARTE3, allora si potrà esportare l'intera tabella oppure le sue porzioni.

Per esportare l'intera tabella, si utilizzi il parametro tables di Export:

```
exp system/manager FILE=expdat.dmp TABLES=(Thumper.VENDITE)
```

Per esportare una porzione o una porzione secondaria specifica, si elenchi il nome della porzione o della porzione secondaria dopo il nome della tabella. Il nome della tabella e quello della porzione dovrebbero essere separati da un segno di due punti (:). Nel listato seguente, viene esportata la porzione PARTE1:

```
exp system/manager FILE=expdat.dmp TABLES=(Thumper.VENDITE:Parte1)
```

Se si esporta una porzione, verranno esportate anche tutte le sue porzioni secondarie.

## Import

L'utilità Import legge il file di dump di esportazione ed esegue i comandi in esso memorizzati. Import potrebbe servire per restituire in modo selettivo gli oggetti o gli utenti dal file di dump di esportazione.

Import può essere eseguita in modo interattivo oppure tramite i file di comandi. Le opzioni runtime che possono essere specificate per Import e i loro valori predefiniti sono elencate nella Tabella 40.2.

Alcuni parametri di Import sono in conflitto reciproco oppure potrebbero portare a istruzioni incoerenti per Import. Per esempio, le impostazioni full=Y e owner=HR fallirebbero poiché il parametro full chiama un'importazione Full mentre il parametro owner chiama un'importazione User.

### Requisiti dei segmenti di undo

Per default, un database eseguirà un commit dopo la completa importazione di ogni tabella. Di conseguenza, il segmento di rollback conterrà un RowID per ogni riga importata. Per ridurre le dimensioni delle voci dei segmenti di undo, si specifichi commit=Y insieme a un valore per buffer. Dopo i dati di ogni buffer verrà eseguito un commit, come mostrato nell'esempio seguente. Nel primo comando import mostrato, un commit viene eseguito dopo il caricamento di ogni tabella. Nel secondo comando mostrato, un commit viene eseguito ogni volta che si inseriscono 64.000 byte di dati.

```
imp system/manager file=expdat.dmp  
imp system/manager file=expdat.dmp buffer=64000 commit=Y
```

Quanto dovrebbe essere grande buffer? La sua dimensione dovrebbe essere sufficiente a contenere la singola riga più grande da importare. Se non si conosce la lunghezza della riga più grande esportata, si incomincia con un valore discreto (come 50.000) e si esegua l'importazione. Se si riceve un errore IMP-00020, significa che la dimensione del buffer non è sufficiente. La si aumenti e si esegua nuovamente l'importazione.

Quando si usa commit=Y, si dovrà tenere a mente che per ogni array del buffer viene eseguito un commit. Questo implica che, nel caso in cui l'importazione di una tabella fallisca, è possibile

**Tabella 40.2** Parametri di Import e valori predefiniti.

PAROLA RISERVATA	DESCRIZIONE
userid	Nome utente/password dell'account che esegue l'importazione. Dev'essere il primo parametro e il testo "userid" è facoltativo.
buffer	Dimensioni del buffer utilizzato per prelevare le righe di dati. Il valore standard dipende dal sistema; di solito a questo parametro viene assegnato un valore alto (maggiori di 100.000).
file	Nome del file di dump da importare.
show	Un flag Y/N per specificare se i contenuti del file devono essere solo visualizzati. Il valore predefinito è N.
ignore	Un flag Y/N per indicare se Import deve ignorare gli errori verificatisi durante l'esecuzione dei comandi create. Si utilizza se gli oggetti da importare esistono già. Il valore predefinito è N.
grants	Un flag Y/N per indicare se devono essere importate le concessioni (grant) sugli oggetti del database. Il valore predefinito è Y.
indexes	Un flag Y/N per indicare se devono essere importati gli indici delle tabelle. Il valore predefinito è Y.
rows	Un flag Y/N per indicare se devono essere importate le righe. Se si specifica "N" viene eseguito solo il codice DDL per gli oggetti del database. Il valore predefinito è Y.
log	Nome del file nel quale verrà scritto il log di importazione.
full	Un flag Y/N; se posto a "Y" viene importato l'intero file di dump. Il valore predefinito è N.
fromuser	Un elenco di account del database i cui oggetti devono essere letti dal file di dump (quando FULL = "N").
touser	Un elenco di account del database nei cui oggetti deve essere importato il file di dump. fromuser e touser non devono avere lo stesso valore.
tables	L'elenco delle tabelle da importare. A partire da Oracle9i è accettato l'uso dei caratteri jolly % e _ per i nomi delle tabelle.
recordlength	La lunghezza in byte del record del file di dump. Di solito si utilizza il valore predefinito, a meno che non si debba trasferire il file da un sistema operativo a un altro.
commit	Un flag Y/N per indicare se Import deve eseguire un comando commit dopo il caricamento di ogni array (le cui dimensioni sono specificate in BUFFER). Se viene impostato a "N", il comando commit viene impartito dopo ogni tabella. Per tabelle di grandi dimensioni, commit=N richiede l'uso di segmenti di undo altrettanto grandi.
parfile	Il nome di un file di parametri da passare a Import. Questo file può contenere tutti i parametri elencati in questa tabella.
constraints	Un flag Y/N che indica se devono essere importati anche i vincoli presenti nelle tabelle. Il valore predefinito è Y.
destroy	Un flag Y/N per indicare se devono essere eseguiti i comandi create tablespace che si trovano nei file dump di esportazioni complete (distruggendo così i file di dati del database di destinazione). Il valore predefinito è N.
indexfile	Questa opzione scrive tutti i comandi create table, create cluster e create index su un file invece di eseguirli. Tutti i comandi tranne create index vengono trasformati in commenti. Se è stato impostato constraint=Y, nel file verranno scritti anche i vincoli. Questo file può poi essere eseguito (con lievi modifiche) dopo un'importazione con indexes=N. È molto utile per suddividere le tabelle e gli indici in tablespace separate.

(segue)

**Tabella 40.2** Parametri di Import e valori predefiniti. (*continua*)

PAROLA RISERVATA	DESCRIZIONE
skip_unusable_indexes	Un flag Y/N che indica se Import deve tralasciare gli indici di partizioni contrassegnate come "unusable". In genere, per migliorare le prestazioni, si preferisce ignorare gli indici durante l'importazione e ricrearli dopo manualmente. Il valore predefinito è N.
feedback	Il numero di righe dopo cui visualizzare la progressione dell'importazione delle tabelle. Il valore standard è 0, che non mostra alcuna informazione finché una tabella non sia stata completamente importata.
toid_novalidate	Consente a Import di saltare la convalida di determinati tipi di oggetti.
filesize	Il valore massimo di dimensione del file di dump che è stato specificato in esportazione.
statistics	Un flag per indicare se devono essere importate le statistiche precalcolate. Il valore predefinito è ALWAYS; gli altri valori sono NONE, SAFE (per le statistiche non discutibili) e RECALCULATE (ricalcola durante l'importazione).
resumable	Un flag Y/N per indicare se la sessione può essere recuperata in seguito a degli errori. Il valore predefinito è N.
resumable_name	Il valore specificato viene inserito nella vista DBA_RESUMABLE per aiutare a identificare l'istruzione recuperabile.
resumable_timeout	Il tempo di attesa per le istruzioni recuperabili.
compile	Un flag Y/N per indicare se procedure, funzioni e package devono essere ricompilati durante l'importazione. Il valore predefinito è Y.
volsize	Numero di byte da scrivere per ciascun volume del nastro.
transport_tablespace	Un flag Y/N per indicare che nel database devono essere importati i metadati della tablespace trasportabili. Il valore predefinito è N.
tablespaces	Il nome o l'elenco delle tablespace che devono essere trasportate nel database.
datafiles	L'elenco dei file di dati che devono essere trasportati nel database.
ts_owner	L'elenco dei proprietari dei dati contenuti nella tablespace trasportabile.

che alcune righe in quella tabella siano già state importate e che il loro commit sia già stato effettuato. Quindi, il caricamento parziale potrebbe essere utilizzato o cancellato prima di eseguire nuovamente l'importazione.

Nelle tabelle con le colonne BFILE, LONG, LOB, REF, ROWID e UROWID, le righe vengono inserite separatamente. Il valore del buffer non deve soddisfare le porzioni LONG e LOB della riga, Import cercherà di allocare altre aree di buffer in base alle esigenze di queste porzioni.

### Importazione in account differenti

Per spostare gli oggetti da un utente all'altro tramite Export/Import, si esegua un'esportazione User del proprietario degli oggetti. Durante l'importazione, si specifichi il proprietario come fromuser e l'account che possiederà questi oggetti come touser.

Per esempio, per copiare gli oggetti di THUMPER nell'account FLOWER si potrebbero eseguire i comandi seguenti. Il primo comando esporta il proprietario THUMPER mentre il secondo comando importa gli oggetti di THUMPER nell'account FLOWER.

```
exp system/manager file=thumper.dat owner=thumper grants=N
indexes=Y compress=Y rows=Y

imp system/manager file=thumper.dat FROMUSER=thumper TOUSER=flower
rows=Y indexes=Y
```

Per ulteriori dettagli su Export e Import, insieme a esempi sul recupero dei dati, fare riferimento alla *Oracle9i Utilities Guide*.

## Backup fuori linea

Un backup fuori linea (chiamato in qualche caso anche backup a freddo) è un backup fisico dei file di database, eseguito dopo che il database è stato chiuso con una chiusura normale, immediata, oppure con uno shutdown transactional. Mentre il database è chiuso, per ciascuno dei file utilizzati attivamente dal database viene effettuato un backup. Questi file offrono un'immagine completa del database così come era nel momento in cui è stato chiuso.

**NOTA** *Non si dovrebbe fare affidamento su un backup fuori linea eseguito dopo una chiusura con interruzione, poiché potrebbe essere incoerente. Se si deve eseguire una chiusura interrotta, è consigliabile avviare di nuovo il database ed eseguire una chiusura normale, una chiusura immediata oppure uno shutdown transactional prima di avviare il backup fuori linea.*

Durante i backup a freddo si dovrebbe effettuare il backup per i seguenti file:

- Tutti i file di dati.
- Tutti i control file.
- Tutti i redo log in linea.

Si potrebbe decidere di eseguire il backup anche per il file dei parametri di inizializzazione del database, in particolar modo se il backup fungerà da base per un processo di recupero in seguito a disastri.

L'esecuzione del backup per tutti questi file *mentre il database è chiuso* fornisce un'immagine completa del database così come era nel momento in cui è stato chiuso. Più avanti si potrebbe recuperare l'intero gruppo di questi file dai backup e il database sarebbe in grado di funzionare. Non è valido eseguire un backup di file system del database mentre questo è aperto a meno che non si esegua un backup in linea (come discusso più avanti in questo capitolo).

Idealmente, tutti i file di dati si trovano in directory allo stesso livello su ciascun dispositivo. Per esempio, tutti i file di database potrebbero essere memorizzati in una sottodirectory specifica dell'istanza sotto una directory /oracle per ogni dispositivo (come /db01/oracle/MYDB). Directory come queste dovrebbero comprendere tutti i file di dati, i redo log file e i control file per un database. L'unico file che si potrebbe aggiungere al backup fuori linea, che non si troverà in questa ubicazione, è il file dei parametri di inizializzazione di produzione, che dovrebbe trovarsi nella sottodirectory /app/oracle/admin/INSTANCE\_NAME/pfile sotto la directory base del software Oracle, oppure nella directory /database sotto la directory home del software Oracle.

Se nell'esempio precedente si utilizza la struttura delle directory, i comandi di backup saranno semplificati moltissimo, dal momento che si potranno usare i caratteri jolly nei nomi file. Dopo aver chiuso il database, si esegua il backup per i file nell'area di destinazione del backup (un nastro oppure un'area separata del disco).

**NOTA** Se necessario, è possibile eseguire contemporaneamente un backup per i file *init.ora* e *config.ora*.

Dato che i backup fuori linea comportano modifiche alla disponibilità del database, solitamente vengono programmati per aver luogo di notte.

I backup fuori linea sono molto affidabili. Per ridurre l'impatto che esercitano sulla disponibilità del database, si potrebbero utilizzare i backup in linea. Come descritto nel prossimo paragrafo, i backup in linea utilizzano la modalità ARCHIVELOG di Oracle per consentire backup di file system coerenti durante l'utilizzo del database.

## Backup in linea

I backup in linea possono essere utilizzati per qualsiasi database eseguito in modalità ARCHIVELOG. In questa modalità, vengono archiviati i redo log in linea creando un log completo di tutte le transazioni all'interno del database.

Oracle scrive nei redo log file in linea in modo ciclico; dopo aver riempito il primo file di log, Oracle inizia a scrivere nel secondo log fino a quando non lo ha riempito, quindi inizia a scrivere nel terzo. Una volta riempito l'ultimo redo log file in linea, il processo di background LGWR (Log Writer) inizia a sovrascrivere il contenuto del primo redo log file.

Quando Oracle viene eseguito in modalità ARCHIVELOG, il processo di background ARC0-ARC9 (Archiver) crea una copia per ogni redo log file prima di sovrascriverlo. Solitamente, questi redo log file archiviati vengono scritti in una periferica di disco. I redo log file archiviati potrebbero essere scritti direttamente in una periferica a nastro, tuttavia questa operazione tende a essere molto impegnativa.

È possibile eseguire i backup di file system di un database mentre questo database è aperto, a condizione che il database venga eseguito in modalità ARCHIVELOG. Un backup in linea richiede l'impostazione di ogni tablespace in uno stato di backup, l'esecuzione del backup dei suoi file di dati e infine il ripristino della tablespace nel suo stato normale.

**NOTA** Quando si utilizza l'utility RMAN fornita da Oracle, non è necessario inserire ogni tablespace in uno stato di backup. L'utility inserirà ed estrarrà automaticamente la tablespace dallo stato di backup.

Il database può essere recuperato completamente da un backup in linea e, tramite i redo log archiviati, il rollforward può essere effettuato in un qualsiasi momento. Quando si aprirà il database, le transazioni eseguite presenti nel database in quel momento saranno state recuperate e il rollback delle transazioni non eseguite sarà già stato effettuato.

Mentre il database è aperto, viene effettuato il backup per i file seguenti:

- Tutti i file di dati.
- Tutti i redo log file archiviati.
- Un control file, con il comando *alter database*.

Le procedure dei backup in linea sono molto potenti per due motivi. In primo luogo, forniscono un recupero puntuale completo. I database non eseguiti in modalità ARCHIVELOG potranno essere recuperati solamente nel momento in cui avviene il backup. In secondo luogo, permettono al database di rimanere aperto durante il backup di file system. Quindi, persino i database che non possono essere chiusi per requisiti dell'utente potranno comunque avere dei backup di file system.

## Primi passi

Per sfruttare la capacità di ARCHIVELOG, per prima cosa si dovrà collocare il database in modalità ARCHIVELOG. Il listato seguente mostra i passaggi necessari per mettere un database in modalità ARCHIVELOG. Si esegua SQLPLUS e si esegua anche il MOUNT del database (in questi esempi si inserisca il suo nome al posto di “mydb”), infine lo si modifichi come mostrato di seguito:

```
SQL> connect system/manager as sysdba
SQL> startup mount mydb;
SQL> alter database archivelog;
SQL> alter database open;
```

Il comando seguente visualizzerà lo stato ARCHIVELOG corrente del database dall’intero di SQLPLUS:

```
archive log list
```

**NOTA** *Per visualizzare il redo log in linea attualmente attivo e il suo numero di sequenza, si esegua una query sulla vista dinamica V\$LOG.*

Per riportare un database in modalità NOARCHIVELOG, si utilizzi il seguente gruppo di comandi dopo aver chiuso il database:

```
SQL> connect system/manager as sysdba
SQL> startup mount mydb;
SQL> alter database noarchivelog;
SQL> alter database open;
```

Un database collocato in modalità ARCHIVELOG rimarrà in questa modalità fino a quando non verrà impostato in modalità NOARCHIVELOG. La posizione dei redo log file archiviati è determinata dalle impostazioni nel file dei parametri del database. I parametri importanti in Oracle9i sono i seguenti (con valori di esempio):

```
log_archive_dest_1      = /db01/oracle/arch/CC1/arch
log_archive_dest_state_1 = ENABLE
log_archive_start        = TRUE
log_archive_format       = arch%$.arc
```

In questo esempio, i redo log file archiviati sono stati scritti nella directory /db01/oracle/arch/CC1. I redo log file archiviati inizieranno tutti con le lettere “arch,” seguite da un numero di sequenza. Per esempio, la directory dei redo log file archiviati potrebbe contenere i file seguenti:

```
arch_170.arc
arch_171.arc
arch_172.arc
```

Ciascuno di questi file contiene i dati tratti da un solo redo log in linea. Questi sono numerati in modo sequenziale, nell’ordine in cui sono stati creati. La dimensione dei redo log file archiviati varia, tuttavia non supera la dimensione dei redo log file in linea.

Se la directory di destinazione dei redo log file archiviati esaurisce lo spazio, ARCH terminerà l’elaborazione dei dati redo log e il database rimarrà temporaneamente in attesa. Questa situazione può essere risolta aggiungendo più spazio al disco di destinazione dei redo log file

archiviati oppure eseguendo il backup dei redo log file archiviati rimuovendoli poi da questa directory.

**NOTA** *I redo log file non devono mai essere eliminati fino a quando non si sarà eseguito un loro backup e non si sarà verificata la possibilità di recuperarli correttamente.*

Anche se il parametro LOG\_ARCHIVE\_START del parametro di inizializzazione può essere impostato a TRUE, il database *non* si troverà in modalità ARCHIVELOG a meno che il comando alter database archivelog mostrato in precedenza non sia stato eseguito. Una volta in modalità ARCHIVELOG, il database rimarrà in questa modalità durante i successivi avvii e chiusure di database fino a quando non lo si inserirà esplicitamente in modalità NOARCHIVELOG con il comando alter database noarchivelog.

### Esecuzione dei backup di database in linea

Non appena un database viene eseguito in modalità ARCHIVELOG, si potrà effettuarne il backup mentre è aperto e disponibile per gli utenti. Questa capacità permette di ottenere una disponibilità ventiquattro ore su ventiquattro del database garantendone sempre il recupero.

Anche se i backup in linea possono essere effettuati durante le normali ore di lavoro, per diversi motivi è consigliabile programmarli negli orari di minor attività degli utenti. Innanzitutto, i backup in linea utilizzeranno i comandi del sistema operativo per eseguire il backup dei fili fisici, e questi comandi useranno le risorse di I/O disponibili nel sistema (influendo così sulle prestazioni del sistema per gli utenti interattivi). In secondo luogo, mentre viene eseguito il backup delle tablespace, cambia il modo in cui le transazioni vengono scritte nei redo log file archiviati. Quando si inserisce una tablespace in modalità “backup”, il processo DBWR scrive tutti i blocchi nella buffer cache che appartiene a qualsiasi file faccia parte della tablespace riportata nel disco. Quando i blocchi vengono riletti nella memoria e poi modificati, la prima volta che li si modifica questi verranno copiati nel log buffer. Fino a quando rimangono nella buffer cache, non verranno copiati nuovamente nel redo log file in linea. Quest’ultimo occuperà molto più spazio nella directory di destinazione del redo log file archiviato.

Il file dei comandi per un backup a caldo è composto da tre parti.

1. Un backup per singola tablespace dei file di dati, che a sua volta è costituito da:
  - a. impostazione della tablespace nello stato di backup;
  - b. esecuzione del backup dei file di dati della tablespace;
  - c. ripristino della tablespace nel suo stato normale.
2. Backup dei redo log file archiviati, che è costituito da:
  - a. registrazione dei file contenuti nella directory di destinazione del redo log archiviato;
  - b. backup dei redo log file archiviati, quindi (facoltativa) cancellazione o compressione dei medesimi.
3. Backup del control file con il comando alter database backup controlfile.

**NOTA** *Il processo di backup in linea viene automatizzato con l’utility RMAN.*

Per eseguire i backup è consigliabile creare uno script, che dovrebbe essere eseguito a livello del sistema operativo con i comandi SQLPLUS eseguiti per i passaggi 1a, 1c e 3.

Quando si esegue il backup dei file di dati, è possibile effettuarlo direttamente nel nastro o nel disco. Se lo spazio libero su disco è sufficiente, è consigliabile scegliere la seconda opzione, in quanto ridurrà moltissimo il tempo necessario a completare le procedure di backup.

## Recovery Manager

A partire da Oracle8.0 viene fornito un toolset Recovery Manager, chiamato anche RMAN, per permettere agli utenti di eseguire il backup e di recuperare i database in modo automatizzato utilizzando la modalità a linea di comando oppure Recovery Manager da Oracle Enterprise Manager. Per eseguire il backup, ripristinare e recuperare i file di database si possono usare entrambi gli approcci.

Recovery Manager tiene traccia dei backup tramite un Recovery Catalog oppure inserendo le informazioni necessarie nel control file del database per il quale si esegue il backup. Recovery Manager aggiunge nuove capacità di backup che non sono disponibili nelle altre utility di backup di Oracle. Recovery Manager contiene quattro componenti: l'eseguibile RMAN, uno o più database di destinazione, il database Recovery Catalog e il software di gestione Media. Gli unici componenti necessari sono l'eseguibile RMAN e un database di destinazione. Dato che RMAN memorizza automaticamente i suoi metadati nel control file del database di destinazione, non è necessario possedere un Recovery Catalog.

La possibilità di eseguire backup fisici incrementali dei file di dati è senza alcun dubbio la più importante tra le nuove capacità introdotte con Recovery Manager. Durante il backup completo di un file di dati (chiamato *livello 0*), viene eseguito il backup di tutti i blocchi utilizzati nel file dati. Durante il backup cumulativo di un file di dati (livello 1), viene eseguito il backup di tutti i blocchi che sono stati utilizzati dall'ultimo backup completo del file di dati. Il backup incrementale di un file di dati (livello 2) esegue il backup solo di quei blocchi che sono cambiati dall'ultimo backup cumulativo o completo.

La possibilità di effettuare backup incrementali e cumulativi dei file di dati può migliorare significativamente le prestazioni dei backup. I miglioramenti maggiori nelle prestazioni li otterranno i database di grandi dimensioni in cui solamente un piccolo sottoinsieme di una grande tablespace cambia. Con i metodi di backup tradizionali, si dovrebbe eseguire il backup di tutti i file di dati contenuti nella tablespace, mentre con Recovery Manager si esegue il backup dei blocchi che sono cambiati rispetto l'ultimo backup.

Durante il recupero del database con Recovery Manager, è necessario sapere quali file sono correnti, quali ripristinati e il metodo di backup che si intende utilizzare. Se si usa il Recovery Catalog, Recovery Manager memorizza il suo catalogo di informazioni in un database Oracle, quindi si dovrà eseguire il backup di questo database, altrimenti si potrebbe correre il rischio di perdere l'intero catalogo di informazioni relative al backup e al recupero.

Recovery Manager è utilizzato solamente dai DBA, che devono prendere delle decisioni relative all'architettura di Recovery Manager per l'ambiente (per esempio, decidere l'ubicazione del Recovery Catalog). Per comprendere le opzioni di recupero usate e le implicazioni che esse hanno sulla disponibilità di database e il tempo di recupero, si dovrebbe lavorare con il proprio DBA.

## Esecuzione di un backup con Oracle Enterprise Manager (OEM)

Per eseguire backup di qualsiasi livello con lo strumento OEM, è necessario connettersi alla console Server Manager di OEM, selezionare il database appropriato, quindi fare clic con il pulsante destro del mouse per visualizzare le opzioni di database. Dalle opzioni di database, si seleziona l'opzione Backup dal menu Backup Manager. Backup Wizard verrà avviato.

Per effettuare un'operazione di backup, il database di destinazione dev'essere in esecuzione e disponibile. Dopo la prima schermata iniziale, verrà richiesto di selezionare un'opzione di strategia. Si può selezionare una strategia di backup predefinita, oppure personalizzarne una propria. OEM visualizza la schermata delle opzioni Backup Frequency con tre opzioni:

1. Un Decision Support System (DSS) con una frequenza di backup di una volta alla settimana.
2. Un sistema moderatamente aggiornato (OLTP) non molto grande e con una frequenza di backup giornaliera.

3. Un database medio-grande aggiornato spesso, con una frequenza di backup settimanale per i backup completi e notturna per i backup incrementali

L'opzione predefinita è un sistema DSS per il quale il backup viene effettuato una volta la settimana di domenica. Dopo aver selezionato una strategia, verrà richiesto il tempo per eseguire il backup e il database per il quale eseguire il backup. Per un backup immediato, si può scegliere l'opzione Customize dalla schermata Strategy iniziale.

## 40.7 Riepilogo

In questo capitolo, è stata presentata un'introduzione esauriente degli argomenti che i DBA affrontano ogni giorno. Oltre agli argomenti trattati in questa sede, esistono i database di controllo dei DBA, la messa a punto dei database, l'installazione del software, la gestione della connettività di database e molti altri ancora. Per ulteriori informazioni su queste attività, fare riferimento alla documentazione Oracle. Maggiori sono le conoscenze degli sviluppatori sull'amministrazione del database, maggiori sono le probabilità di sviluppare applicazioni che sfruttano le capacità inerenti del database.

## Capitolo 41

# Guida a XML in Oracle

- 41.1 **Document Type Definition, elementi e attributi**
- 41.2 **XML Schema**
- 41.3 **Uso di XSU per selezionare, inserire, aggiornare e rimuovere valori XML**
- 41.4 **Uso di XMLType**
- 41.5 **Altre caratteristiche**

**X**ML (eXtensible Markup Language) è una sintassi standardizzata utilizzata per descrivere dati gerarchici. Più che un linguaggio di programmazione, XML è un formato universale per documenti e dati strutturati. A chi conosce HTML (HyperText Markup Language) risulterà certo familiare la sintassi di XML basata sulle tag, che conferisce a questo linguaggio la flessibilità necessaria a gestire dati complessi. Mentre HTML comunica a un browser Web come presentare i dati, XML comunica alle varie applicazioni il significato dei dati.

XML è adatto a risolvere problemi legati allo scambio dei dati tra sistemi eterogenei; grazie a esso, l'accesso, la conversione e la memorizzazione dei dati contenuti nel database sono operazioni molto semplici. Questo capitolo offrirà una panoramica dei metodi di accesso XML disponibili in Oracle9i. XML e i suoi toolset sono in continua evoluzione, quindi per le informazioni più recenti si consiglia di visitare il sito <http://www.w3.org>.

**NOTA** *Si ringrazia Mike Holder della società TUSC per i suoi contributi a questo capitolo.*

## 41.1 Document Type Definition, elementi e attributi

In XML, le tag o gli “elementi” specifici di un’applicazione sono disposti intorno ai dati che descrivono. La sintassi di queste tag è definita in un tipo speciale di documento XML che prende il nome di Document Type Definition (DTD). La DTD definisce la struttura di un documento XML valido; tale struttura è strettamente gerarchica.

Ogni documento XML è un’istanza di un “gruppo di informazioni”, il modello di dati astratto costituito da un documento e dai suoi elementi informativi. Questi elementi informativi sono principalmente elementi, attributi e contenuto. Per esempio, un gruppo di informazioni XML potrebbe contenere quanto segue:

```
<libro>
  <titolo>MY LEDGER</titolo>
  <capitolo num="1">
    <titolo>Beginning</titolo>
    <texto>&chapter1;/testo>
  </capitolo>
</libro>
```

In questo esempio, le tag <libro> e </libro> definiscono i punti iniziale e finale del gruppo di informazioni. Il documento XML descrive un libro intitolato “MY LEDGER”. Questo libro ha dei capitoli, e il titolo del primo capitolo è “Beginning”. Il testo del capitolo non è memorizzato nel documento XML, in quanto viene creato un puntatore a un file esterno.

Oltre a essere una tag nel file, “titolo” è anche un elemento. Ha del contenuto, in questo caso si tratta di “MY LEDGER”. Come mostrato in questo esempio, è possibile impostare il contenuto degli elementi inserendo dei valori tra le tag (per il titolo), oppure nelle tag (come nell’impostazione del numero di capitolo). Gli elementi possono avere del contenuto, degli elementi figli o entrambi, oppure potrebbero essere vuoti. Per lo scambio di dati, è consigliabile evitare di creare elementi che contengono sia il contenuto sia gli elementi figli. Per creare un elemento vuoto, la sua tag dovrebbe essere nel formato seguente:

```
<nome/>
```

**NOTA** *I nomi degli elementi distinguono tra maiuscole e minuscole.*

Ogni elemento ha un solo padre, mentre un padre può avere un numero qualsiasi di figli. La rigida natura gerarchica di XML prevede che se un elemento inizia all’interno di un altro elemento (come un “capitolo” in un “libro”), la sua tag finale dovrà precedere la tag finale dell’elemento padre.

Come mostrato nell’esempio del libro, gli elementi possiedono degli attributi. Per esempio, l’elemento capitolo ha un attributo di nome *num*. Il valore dell’attributo è racchiuso tra virgolette:

```
<chapter num="1">
```

**NOTA** *Come i nomi degli elementi, anche i nomi degli attributi distinguono tra maiuscole e minuscole.*

Generalmente, le applicazioni utilizzano gli attributi dell’elemento per elementi specifici (come il numero di capitolo) e il contenuto dell’elemento per il grosso dei dati (in questo caso, il testo del capitolo).

Si dice che un documento XML è ben formato se soddisfa le condizioni seguenti.

- Ogni tag iniziale ha una tag finale corrispondente.
- Gli elementi non si sovrappongono.
- Esiste un elemento principale.
- I valori dell’attributo sono sempre racchiusi tra virgolette.
- Un elemento non può avere due attributi con lo stesso nome.
- I commenti e le istruzioni di elaborazione non compaiono nelle tag.
- Nei dati carattere di un elemento o di un attributo non ci sono segni < o & senza escape.

Queste sono le regole di sintassi basilari. Tuttavia, anche se un documento XML rispetta queste regole, potrebbe comunque non essere valido. Per essere valido, il documento XML deve infatti rispettare le regole di una Document Type Definition (DTD). La DTD è un modo formale di descrivere quali elementi e entità potrebbero comparire in un documento XML, e quali sono i contenuti e gli attributi di ciascun elemento.

Per esempio, si consideri il seguente documento XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CustomerList SYSTEM "Example1.dtd">
```

```
<ElencoClienti>
  <Cliente clientePreferito="">
    <ID12345/ID>
    <Indirizzo>
      <Nome>TUSC</Nome>
      <Viat>377 E BUTTERFIELD Rda</Viat>
      <Citta>LOMBARD</Citta>
      <Stato>IL</Stato>
      <Paese>USA</Paese>
      <CAP>60148</CAP>
      <TELEFONO>630-960-2909</TELEFONO>
    </Indirizzo>
    <ValutazioneCredito>EXCELLENT</ValutazioneCredito>
    <IDRepVendite>/<IDRepVen>
    <IDRegione>5/<IDRegione>
    <Commenti>QUESTO È UN DOCUMENTO XML DI PROVA</Commenti>
    <MetodoConsegna>M</MetodoConsegna>
  </Cliente>
</ElencoClienti>
```

La prima linea è una dichiarazione XML, che inizia con “?” e termina con “?“:

```
?xml version="1.0" encoding="UTF-8"?
```

Per ora, l’attributo *version* dovrebbe essere impostato a 1.0. L’attributo *encoding* è facoltativo; il suo valore predefinito è UTF-8. Si può impostare anche un terzo attributo, *standalone*. In questo caso, il documento XML non è indipendente; ha una DTD correlata, come specificato nella seconda linea:

```
!DOCTYPE CustomerList SYSTEM "Example1.dtd"
```

Il file della DTD correlata Example1.dtd per questo esempio è visibile nel listato seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT ElencoClienti (Cliente*)>
<!ELEMENT Cliente (ID, Indirizzo, ValutazioneCredito*, IDRepVendite*,
                   IDRegione, Commenti*, MetodoSpedizione)>
<!ATTLIST Cliente clientePreferito CDATA #IMPLIED>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT Indirizzo (Nome, Via*, Citta, Contea,
                      Stato, Paese, CAP, TELEFONO)>
<!ELEMENT Nome (#PCDATA)>
<!ELEMENT Via (#PCDATA)>
<!ELEMENT Citta (#PCDATA)>
<!ELEMENT Contea (#PCDATA)>
<!ELEMENT Stato (#PCDATA)>
<!ELEMENT Paese (#PCDATA)>
<!ELEMENT CAP (#PCDATA)>
<!ELEMENT TELEFONO (#PCDATA)>
<!ELEMENT ValutazioneCredito (#PCDATA)>
<!ELEMENT IDRepVendite (#PCDATA)>
<!ELEMENT IDRegione (#PCDATA)>
<!ELEMENT Commenti (#PCDATA)>
<!ELEMENT MetodoSpedizione (#PCDATA)>
```

Una DTD è simile alle specifiche del control file per le tabelle esterne. Essa infatti comunica alle applicazioni cosa troveranno nel documento XML, i nomi degli elementi e la loro gerarchia. Un documento XML può avere solamente una DTD.

La sintassi per le dichiarazioni degli elementi è la seguente:

```
<!ELEMENT nome_elemento (modello_contenuto)>
```

in cui *modello\_contenuto* specifica quali figli può o deve possedere un elemento e l'ordine in cui compaiono. Il modello di contenuto più semplice è #PCDATA, il quale indica che l'elemento può contenere solo dati carattere analizzati. Per esempio:

```
<!ELEMENT Nome (#PCDATA)>
```

In questo esempio, l'elemento ElencoClienti contiene nessuno o più elementi “Cliente”, come indicato dal suffisso \* :

```
<!ELEMENT ElencoClienti (Cliente*)>
```

I suffissi ammissibili sono

- \* Permette zero o più occorrenze
- + Permette una o più occorrenze
- ? Permette zero o una occorrenza

Come mostrato nella specifica dell'elemento Indirizzo, è possibile specificare una sequenza di più elementi separandoli con delle virgolette:

```
<!ELEMENT Indirizzo (Nome, Via*, Citta, Contea,  
Stato, Paese, CAP, TELEFONO)>
```

Nella specifica si possono fornire più opzioni, separate dal carattere () barra verticale. Per esempio, si potrebbe specificare un punto in un grafico bidimensionale tramite le sue coordinate verticali e orizzontali, oppure tramite la sua distanza e l'angolo dal centro:

```
<!ELEMENT Punto ((x,y) | (distanza, angolo))>
```

Nelle applicazioni reali, le DTD possono diventare in poco tempo molto complesse. Infatti, in una DTD si possono avere liste di attributi, riferimenti esterni, attributi di entità, ID, inclusioni condizionali e altre strutture. Per un riepilogo esauriente sulle strutture e le opzioni delle DTD, si faccia riferimento alla documentazione relativa a XML.

**NOTA** Prima di creare una DTD per un'operazione aziendale standard, si verifichi se ne esiste già una disponibile. Siti Web come <http://www.xml.org/xml/registry.jsp> forniscono molti esempi di DTD. Questi registri sono una risorsa molto utile per la documentazione di DTD standard aziendali e dei loro componenti.

## 41.2 XML Schema

La specifica XML Schema è un'alternativa alle DTD. Include la tipizzazione dei dati, che diventa particolarmente utile quando si utilizzano database relazionali. Mentre le DTD non sono estensibili (in quanto semplici control file), XML Schema è estensibile nei seguenti modi:

- Parti di schemi possono essere riutilizzati in altri schemi.
- Strutture complesse possono essere riutilizzate in altri schemi.
- Si possono creare tipi di dati derivati basandosi su tipi di dati già esistenti.
- Una singola istanza di documento può fare riferimento a più schemi.

In XML Schema, gli elementi vengono dichiarati con `xsd:element`, come mostrato nel listato seguente:

```
<xsd:element name="Data">
```

Il tipo di dato dell'elemento può essere dichiarato tramite l'attributo *type*:

```
<xsd:element name="Data" type="date">
```

Con gli attributi *minOccurs* e *maxOccurs* si possono specificare il numero minimo e massimo di occorrenze per l'elemento:

```
<xsd:element name="Data" type="date" minOccurs="0" maxOccurs="1">
```

I tipi incorporati XML Schema includono tipi primitivi (come stringhe, valori booleani, numeri, valori in virgola mobile, data, ora e data/ora) e tipi derivati (come `normalizedString`, valori interi, token e linguaggio). Al sito <http://www.w3.org/TR/xmlschema-2/> si può trovare l'elenco completo dei tipi.

Si possono creare e nominare tipi complessi, così come si possono creare tipi di dati astratti in Oracle. Il codice seguente crea il tipo “Indirizzo”:

```
<xsd:complexType name="Indirizzo">
  <xsd:sequenze>
    <xsd:element name="Via1"/>
    <xsd:element name="Via2"/>
    <xsd:element name="Citta"/>
    <xsd:element name="Stato"/>
    <xsd:element name="CAP"/>
  </xsd:sequenze>
</xsd:complexType>
```

Successivamente, si potrà fare riferimento al tipo complesso “Indirizzo” mentre si definiscono gli elementi:

```
<xsd:element name="Elencoindirizzi" type="Indirizzo"/>
<xsd:element name="Indirizzofatturazione" type="Indirizzo"/>
```

Se si dichiara un modello di contenuto per un solo elemento, si utilizzi un tipo anonimo, come mostrato nel listato seguente:

```
<xsd:element name="Nome">
  <xsd:complexType>
    <xsd:sequenze>
      <xsd:element name="Primo"/>
      <xsd:element name="InizialeIntermedia"/>
        <xsd:element name="Ultimo"/>
    </xsd:sequenze>
  </xsd:complexType>
</xsd:element>
```

Gli attributi vengono dichiarati con `xsd:attribute`, come mostrato nell'esempio seguente:

```
<xsd:attribute name="pagato" type="boolean"/>
```

Si possono creare attributi in un tipo complesso:

```
<xsd:element name="Fattura">
  <xsd:complexType>
    <xsd:attribute name="pagato" use="facoltativo default="True"/>
  </xsd:complexType>
</xsd:element>
```

Oltre a creare gli attributi, si possono generare anche gruppi di attributi e gruppi di modelli. I gruppi di modelli sono gruppi o definizioni di elementi, chiamati per essere riutilizzati. L'elemento "gruppo" dev'essere il componente di uno schema di livello superiore (un figlio dell'elemento "schema"). Per esempio, il listato seguente mostra la creazione di un gruppo `Menu`, cui fa riferimento l'elemento "CenaAereo".

```
<xsd:element name="CenaAereo">
  <xsd:sequenze>
    <xsd:element ref="Nome"/>
    <xsd:group ref="Menu"/>
  </xsd:sequenze>
</xsd:element>

<xsd:group name="Menu">
  <xsd:choice>
    <xsd:element name="Pesce" type="Pasto"/>
    <xsd:element name="Carne" type="Pasto"/>
    <xsd:element name="Vegetariano" type="Pasto"/>
  </xsd:choice>
</xsd:group>
```

In genere gli attributi sono raggruppati insieme se gli elementi utilizzano spesso lo stesso insieme di attributi. Ciò assomiglia all'approccio descritto nella Parte quarta di questo libro, in cui gli attributi comuni vengono associati per formare nuovi tipi di dati. Per esempio, la definizione seguente di un elemento "Prodotto" usa un gruppo di attributi che prende il nome di `Dettagliprodotto`. Altri elementi possono riutilizzare questo stesso gruppo di attributi, migliorando così le capacità dell'utente di applicare una rappresentazione standardizzata di questi dati.

```
<xsd:element name="Prodotto">
  <xsd:complexType>
    <xsd:attributeGroup ref="Dettagliprodotto"/>
  </xsd:complexType>
</xsd:element>

<xsd:attributeGroup name="Dettagliprodotto">
  <xsd:attribute name="IDprodotto" use="richiesto" type="xsd:ID"/>
  <xsd:attribute name="nome" use="richiesto" type="xsd:string"/>
  <xsd:attribute name="descrizione" use="richiesto" type="xsd:string"/>
  <xsd:attribute name="unita" type="xsd:positiveInteger"/>
  <xsd:attribute name="prezzo" use="richiesto" type="xsd:decimal"/>
  <xsd:attribute name="scorta" type="xsd:integer"/>
</xsd:attributeGroup>
```

È possibile indurre l'univocità in un'istanza di documento con xsd:unique:

```
<xsd:unique name="IDprodottounico">
  <xsd:selector xpath="tc:Prodotto"/>
  <xsd:field xpath="tc:@IDprodotto"/>
</xsd:unique>
```

Per l'elemento “Prodotto”, l'univocità viene dichiarata allo stesso livello della definizione del tipo complesso. Lo spazio di nomi *tc*: è per lo schema che si crea ed è dichiarato nell'elemento xsd:schema. Il valore xsd:selector specifica gli elementi da vincolare (gli elementi “Prodotto”), mentre xsd:field specifica gli attributi da vincolare (IDprodotto).

xsd:key e xsd:keyref possono essere utilizzati per creare riferimenti tra gli elementi. Per esempio, si può creare un elemento “Cliente” e poi fare riferimento a esso da più elementi “Ricevuta”:

```
<xsd:key name="IDCliente">
  <xsd:selector xpath="tc:DatiVendite"/>
  <xsd:field xpath="tc:Cliente/@IDCliente"/>
</xsd:key>

<xsd:keyref name="articolo" refer="IDcliente">
  <xsd:selector xpath="tc:DatiVendite"/>
  <<xsd:field xpath="tc:Fattura/@IDClienti"/>
</xsd:keyref>
```

Per permettere a un elemento di essere NULL, occorre utilizzare l'attributo *nillable*:

```
<xsd:element name="CenaAerea" nillable="true"/>
```

### 41.3 Uso di XSU per selezionare, inserire, aggiornare e rimuovere valori XML

È possibile usare la XML-SQL Utility (XSU) di Oracle per convertire il risultato di una query SQL in XML. XSU supporta la creazione dinamica di DTD e l'inserimento di codice XML nelle tabelle di database. Inoltre, XSU può servire per aggiornare o rimuovere righe dagli oggetti di database.

Con le API (interfacce per la programmazione di applicazioni) Java e PL/SQL è possibile accedere a XSU. Le classi Java di XSU fanno parte del kernel di Oracle. L'API PL/SQL è un wrapper che pubblica le classi Java in PL/SQL (si veda il Capitolo 36). Dato che le classi XSU sono memorizzate nel database, si potranno scrivere nuove stored procedure per accedere direttamente alle classi Java di XSU. Dal momento che è disponibile un'API PL/SQL, si potrà accedere direttamente alla funzionalità di XSU tramite SQL.

Per esempio, con le procedure PL/SQL di XSU è possibile generare la versione XML della tabella CLASSIFICAZIONE. Il listato seguente crea una procedura di nome PRINTCLOBOUT che verrà chiamata come parte del processo di output dei dati. I dati verranno letti come un CLOB.

```
create or replace procedure PRINTCLOBOUT(result IN OUT NOCOPY CLOB)
  is
xmlstr varchar2(32767);
line varchar2(2000);
```

```

begin
  xmlstr := dbms_lob.SUBSTR(result,32767);
  loop
    exit when xmlstr is null;
    line := substr(xmlstr,1,instr(xmlstr,chr(10))-1);
    dbms_output.put_line('|'||line);
    xmlstr := substr(xmlstr,instr(xmlstr,chr(10))+1);
  end loop;
end;

```

Ora che esiste la procedura PRINTCLOBOUT si esegua il seguente blocco PL/SQL anonimo. Questo blocco effettua la query sulla tabella CLASSIFICAZIONE, quindi questa tabella dovrà essere accessibile dallo schema (si faccia riferimento al CD-ROM allegato):

```

declare
  queryCtx DBMS_XMLQuery.ctxType;
  result CLOB;
begin
  -- configura il contesto di query...
  queryCtx := DBMS_XMLQuery.newContext('select * from classificazione');
  -- ottiene il risultato..
  result := DBMS_XMLQuery.getXML(queryCtx);
  -- Ora, si può utilizzare il risultato per inserirlo nelle tabelle/inviare come messaggi..
  printClobOut(result);
  DBMS_XMLQuery.closeContext(queryCtx); -- si deve chiudere la query..
end;

```

Il risultato è la tabella CLASSIFICAZIONE, formattata con le tag XML:

```

| <?xml version = '1.0'?>
| <ROWSET>
|   <ROW num="1">
|     <CLASSIFICAZIONE>1</CLASSIFICAZIONE>
|     <DESCRIZIONECLASSIFICAZIONE>EVASIONE</DESCRIZIONECLASSIFICAZIONE>
|   </ROW>
|   <ROW num="2">
|     <CLASSIFICAZIONE>2</CLASSIFICAZIONE>
|     <DESCRIZIONECLASSIFICAZIONE>INFORMAZIONI DI BASE</DESCRIZIONECLASSIFICAZIONE>
|   </ROW>
|   <ROW num="3">
|     <CLASSIFICAZIONE>3</CLASSIFICAZIONE>
|     <DESCRIZIONECLASSIFICAZIONE>CONSIGLIATO</DESCRIZIONECLASSIFICAZIONE>
|   </ROW>
|   <ROW num="4">
|     <CLASSIFICAZIONE>4</CLASSIFICAZIONE>
|     <DESCRIZIONECLASSIFICAZIONE>VIVAMENTE CONSIGLIATO</DESCRIZIONECLASSIFICAZIONE>
|   </ROW>
|   <ROW num="5">
|     <CLASSIFICAZIONE>5</CLASSIFICAZIONE>
|     <DESCRIZIONECLASSIFICAZIONE>LETTURA INDISPENSABILE</DESCRIZIONECLASSIFICAZIONE>
|   </ROW>
| </ROWSET>

```

PL/SQL procedure successfully completed.

## Procedure di inserimento, aggiornamento e annullamento con XSU

Per inserire un documento in una tabella o in una vista, si utilizzi il package DBMS\_XMLSave. XSU analizzerà questo documento e creerà un'istruzione insert che inserisce i valori in tutte le colonne della tabella o della vista. Un elemento assente viene trattato come valore NULL.

```
create or replace procedure INSPROC(xmlDoc IN CLOB,
tableName IN VARCHAR2) is
    insCtx DBMS_XMLSave.ctxType;
    rows number;
begin
    insCtx := DBMS_XMLSave.newContext(tableName); -- ottiene il contesto
    rows := DBMS_XMLSave.insertXML(insCtx,xmlDoc); -- inserisce il documento
    DBMS_XMLSave.closeContext(insCtx);           -- chiude l'handle
end;
```

Ora i documenti XML possono essere passati alla procedura INSPROC sotto forma di valori di input CLOB insieme al nome della tabella. INSPROC accetterà il documento XML e inserirà i suoi valori nella tabella. Chiamando una procedura separata di nome UPDATEPROC, mostrata nel listato seguente, si possono effettuare degli aggiornamenti.

```
create or replace procedure UPDATEPROC ( xmlDoc IN clob) is
    updCtx DBMS_XMLSave.ctxType;
    rows number;
begin
    updCtx := DBMS_XMLSave.newContext('CLASSIFICAZIONE'); -- ottiene il contesto
    DBMS_XMLSave.clearUpdateColumnList(updCtx); -- annulla le impostazioni
    DBMS_XMLSave.setKeyColumn(updCtx,'CLASSIFICAZIONE'); -- imposta CLASSIFICAZIONE come chiave
    rows := DBMS_XMLSave.updateXML(updCtx,xmlDoc); -- aggiorna
    DBMS_XMLSave.closeContext(updCtx);           -- chiude il contesto
end;
```

In questo esempio, la colonna Classificazione è la colonna di chiave usata per la clausola where dell'aggiornamento:

```
DBMS_XMLSave.setKeyColumn(updCtx,'CLASSIFICAZIONE'); -- imposta CLASSIFICAZIONE come chiave
```

I valori nuovi vengono passati come un documento XML in formato CLOB, e i valori della tabella CLASSIFICAZIONE vengono aggiornati in base ai contenuti del documento XML.

Le operazioni di cancellazione funzionano in modo molto analogo. La procedura seguente accetta un documento XML come proprio input, imposta la colonna Classificazione come chiave per la tabella CLASSIFICAZIONE ed esegue l'operazione di cancellazione basandosi sui valori chiave contenuti nel documento di input. Si osservi che newContext accetta il nome della tabella CLASSIFICAZIONE come proprio valore di input, mentre setKeyColumn accetta il nome della colonna chiave (che in questo caso è sempre Classificazione).

```
create or replace procedure DELETEPROC(xmlDoc IN clob) is
    delCtx DBMS_XMLSave.ctxType;
    rows number;
begin
    delCtx := DBMS_XMLSave.newContext('CLASSIFICAZIONE');
    DBMS_XMLSave.setKeyColumn(delCtx,'CLASSIFICAZIONE');
```

---

```

rows := DBMS_XMLSave.deleteXML(delCtx,xmlDoc);
DBMS_XMLSave.closeContext(delCtx);
end;

```

## XSU e Java

Se si preferisce, è possibile eseguire le operazioni DML di XSU chiamando delle classi Java al posto delle procedure PL/SQL. Al posto del package DBMS\_XMLQuery, si dovrebbe eseguire la classe oracle.xml.sql.query.OracleXMLQuery. Al posto del package DBMS\_XMLSave, si usi oracle.xml.sql.dml.OracleXMLSave.

Per esempio, il campione di programma Java fornito da Oracle per le query XML è stato leggermente personalizzato nel listato seguente. Esso si basa sulla gestione della connessione eseguita in modo molto simile agli esempi SQLJ e JDBC (si veda il Capitolo 35). Il programma importa la classe oracle.xml.sql.query.OracleXMLQuery, permettendole di ottenere la stringa XML; il metodo System.out.println di Java viene chiamato per visualizzare l'output.

```

import oracle.xml.sql.query.OracleXMLQuery;
import java.lang.*;
import java.sql.*;

// Classe per verificare la generazione della stringa!
class testXMLSQL {

    public static void main(String[] argv)
    {

        try{
            // Stabilisce la connessione
            Connection conn = getConnection("pratica","pratica");

            // Crea la classe di query.
            OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from classificazione");

            // Ottiene la stringa XML
            String str = qry.getXMLString();

            // Visualizza l'output XML
            System.out.println(" L'output XML è:\n"+str);
            // Chiudere sempre la query per eliminare qualsiasi risorsa.
            qry.close();
        }catch(SQLException e){
            System.out.println(e.toString());
        }
    }

    // Ottiene la connessione conoscendo il nome e la password!
    private static Connection getConnection(String username,
    String password)
        throws SQLException
    {
        // regista il driver JDBC..
        DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
    }
}

```

```

// Crea la connessione con il driver OCI8
Connection conn =
DriverManager.getConnection("jdbc:oracle:oci8:@or90",username,password);

return conn;
}
}

```

Per dettagli su SQLJ, sulla gestione delle connessioni e Java, fare riferimento alla Parte quinta di questo libro.

## Personalizzazione del processo di query

Per semplificare l'iterazione con i package DBMS\_XMLQuery e DBMS\_XMLS救人 è possibile creare delle procedure personalizzate. Per esempio, il listato seguente crea una funzione di nome GET\_XML, che selezionerà le righe dalla tabella e la visualizzerà in formato XML.

```

create function GET_XML (in_SQL varchar2)
  return CLOB
is queryCtx DBMS_XMLQuery.ctxType;
  result CLOB;
begin   queryCtx := DBMS_XMLQuery.newContext(in_SQL);
  result := DBMS_XMLQuery.getXML(queryCtx);
  DBMS_XMLQuery.closeContext(queryCtx);
  return result;
end;

```

I dati verranno restituiti in formato CLOB, quindi il comando set long potrà servire per impostare la lunghezza di visualizzazione massima (quella predefinita è di 80 caratteri). È possibile chiamare GET\_XML dall'interno delle query SQL, come mostrato nel listato seguente.

```

set pagesize 100 long 2000

select GET_XML('select * from categoria') from DUAL;

GET_XML('SELECT*FROMCATEGORIA')
-----
<?xml version = '1.0'?>
<ROWSET>
<ROW num="1">
  <NOMECategoria>CONSADULTI</NOMECategoria>
  <CATEGORIAPADRE>ADULTI</CATEGORIAPADRE>
  <SOTTOCATEGORIA>CONSULTAZIONE</SOTTOCATEGORIA>
</ROW>
<ROW num="2">
  <NOMECategoria>NARRADULTI</NOMECategoria>
  <CATEGORIAPADRE>ADULTI</CATEGORIAPADRE>
  <SOTTOCATEGORIA>NARRATIVA</SOTTOCATEGORIA>
</ROW>
<ROW num="3">
  <NOMECategoria>SAGGIADULTI</NOMECategoria>
  <CATEGORIAPADRE>ADULTI</CATEGORIAPADRE>
  <SOTTOCATEGORIA>SAGGI</SOTTOCATEGORIA>
</ROW>

```

```

<ROW num="4">
  <NOMECategoria>ILLUSRAGAZZI</NOMECategoria>
  <CATEGORIAPADRE>RAGAZZI</CATEGORIAPADRE>
  <SOTTOCategoria>ILLUSTRATI</SOTTOCategoria>
</ROW>
<ROW num="5">
  <NOMECategoria>NARRRAGAZZI</NOMECategoria>
  <CATEGORIAPADRE>RAGAZZI</CATEGORIAPADRE>
  <SOTTOCategoria>NARRATIVA</SOTTOCategoria>
</ROW>
<ROW num="6">
  <NOMECategoria>SAGGIRAGAZZI</NOMECategoria>
  <CATEGORIAPADRE>RAGAZZI</CATEGORIAPADRE>
  <SOTTOCategoria>SAGGI</SOTTOCategoria>
</ROW>
</ROWSET>

```

#### 41.4 Uso di XMLType

A partire da Oracle9i è possibile utilizzare XMLType come tipo di dati nelle tabelle. XMLType può servire per memorizzare ed eseguire la query sui dati XML nel database. Come tipo, XMLType ha delle funzioni membro (si veda il Capitolo 30) per accedere, estrarre ed eseguire la query sui dati XML utilizzando una classe di operazioni note come espressioni Xpath. I package SYS\_XMLGEN, SYS\_XMLAGG e DBMS\_XMLGEN creano valori XMLType da dati relazionali a oggetti già esistenti. Quando si determina che una colonna utilizzerà il tipo di dati XMLType, Oracle memorizzerà i dati in un tipo di dati CLOB.

Il listato seguente mostra la creazione di una tabella con il tipo di dati XMLType:

```

create table MIA_TABELLA_XML
(Key1      NUMBER,
Xml_Column  SYS.XMLTYPE);

```

Se si descrive la tabella, compariranno le sue colonne:

```
desc MIA_TABELLA_XML
```

Name	Null?	Type
KEY1		NUMBER
XML_COLUMN		SYS.XMLTYPE

Se si fa una descrizione più approfondita, compariranno le funzioni membro per la colonna XMLType:

```
set describe depth 5
```

Name	Null?	Type
KEY1		NUMBER
XML_COLUMN		SYS.XMLTYPE

## METHOD

-----  
 STATIC FUNCTION CREATEXML RETURNS XMLTYPE  
 Argument name Type In/Out Default?  
 -----  
 XMLDATA CLOB IN

## METHOD

-----  
 STATIC FUNCTION CREATEXML RETURNS XMLTYPE  
 Argument name Type In/Out Default?  
 -----  
 XMLDATA VARCHAR2 IN

## METHOD

-----  
 MEMBER FUNCTION EXTRACT RETURNS XMLTYPE  
 Argument name Type In/Out Default?  
 -----  
 XPATH VARCHAR2 IN

## METHOD

-----  
 MEMBER FUNCTION EXISTSNODE RETURNS NUMBER  
 Argument name Type In/Out Default?  
 -----  
 XPATH VARCHAR2 IN

## METHOD

-----  
 MEMBER FUNCTION ISFRAGMENT RETURNS NUMBER

## METHOD

-----  
 MEMBER FUNCTION GETCLOBVAL RETURNS CLOB

## METHOD

-----  
 MEMBER FUNCTION GETSTRINGVAL RETURNS VARCHAR2

## METHOD

-----  
 MEMBER FUNCTION GETNUMBERVAL RETURNS NUMBER

Con una chiamata al metodo CREATEXML di XMLType si potranno inserire i valori in MIA\_TABELLA\_XML:

```
insert into MIA_TABELLA_XML (Key1, Xml_Column)
values (1, SYS.XMLTYPE.CREATEXML
('<libro>
<titolo>MY LEDGER</titolo>
<capitolo num= "1 ">
<titolo>Beginning</titolo>
<texto>Questo è il registro di G.B. Talbot</testo>
</capitolo>
</libro>'));
```

Con una chiamata al metodo GETCLOBVAL si può eseguire la query sui dati. Questa query

```
select M.Xml_Column.GETCLOBVAL() as XML_Data
  from MIA_TABELLA_XML M
 where Key1 = 1;
```

restituisce i dati seguenti:

```
XML_DATA
-----
<libro>
  <titolo>MY LEDGER</titolo>
  <capitolo num="1">
    <titolo>Beginning</titolo>
    <texto>Questo è il registro di G.B. Talbot</testo>
  </capitolo>
|</libro>
```

**NOTA** È possibile generare indici funzionali su colonne create con il tipo di dati XMLType per migliorare le prestazioni delle query.

## 41.5 Altre caratteristiche

Questo capitolo ha mostrato un’interazione di livello molto elevato tra i dati XML e il database Oracle. Esistono molte altre possibilità, compreso l’uso dei fogli di stile per la formattazione dei dati e la creazione di pagine XSQL. Si possono creare persino indici di testo su colonne XMLType:

```
create an index XML_INDEX on MIA_TABELLA_XML (Xml_Column)
indextype is ctxsys.context;
```

**NOTA** Per ulteriori dettagli sugli indici di testo, fare riferimento al Capitolo 24.

Oracle mette a disposizione altri package per interagire con i dati XML, tra cui:

DBMS_XMLGEN	Converte i risultati delle query SQL in un formato XML canonico, restituendoli come CLOB: DBMS_XMLGEN è implementato in C e compilato nel kernel del database. DBMS_XMLGEN assomiglia, a livello di funzionalità, al package DBMS_XMLQuery.
SYS_XMLGEN	Genera codice XML nelle query SQL. DBMS_XMLGEN e altri package operano a livello di query, dando risultati aggregati per l’intera query. SYS_XMLGEN opera su <i>singolo</i> argomento in una query SQL e converte il risultato in XML. SYS_XMLGEN accetta un valore scalare, un tipo di oggetto oppure un’istanza XMLType da convertire in un documento XML. Accetta anche un oggetto XMLGenFormatType facoltativo, per specificare le opzioni di formattazione per il risultato. SYS_XMLGEN restituisce un tipo XML.
SYS_XMLAGG	Aggrega più set di XMLTypes. SYS_XMLAGG aggrega tutti i documenti/frammenti XML di input e produce un singolo documento XML concatenando frammenti XML e aggiungendo una tag di livello superiore, che ha come impostazione predefinita ROWSET.

Oracle mette a disposizione delle utility per supportare utilizzi aggiuntivi, come oraxml (un’interfaccia a linea di comando usata per analizzare un documento XML) e oraxsl

(un'interfaccia a linea di comando usata per applicare un foglio di stile su più documenti XML). Un'analisi completa di ogni utility e metodo di accesso ai dati andrebbe ben oltre lo scopo di questo libro; per altre opzioni e programmi di esempio, si prega di consultare la *Oracle9i Application Developer's Guide – XML*.



## Capitolo 42

# Guida di riferimento alfabetica

- 42.1 **Contenuti della guida di riferimento**
- 42.2 **Che cosa non contiene la guida di riferimento**
- 42.3 **Formato generale delle voci**
- 42.4 **Ordine delle parole chiave**

Copyright © 2001, 2002, Oracle Corporation. Tutti i diritti riservati.

**Q**uesto capitolo contiene la documentazione per l'utente di Oracle Corporation riprodotta qui con l'autorizzazione di Oracle Corporation e offre un breve riepilogo dei comandi SQL più comuni utilizzati dai DBA. I comandi sono disposti in ordine alfabetico. Questo materiale è estratto dai documenti *Oracle9i SQL Reference*, *SQL\*Plus User's Guide and Reference*, e *Oracle9i Database Utilities Release 1*.

In questo capitolo vengono presentati i riferimenti relativi alla maggior parte dei comandi di Oracle, le parole chiave, le funzionalità e le funzioni, con riferimenti incrociati tra gli argomenti. Questa guida di riferimento è stata studiata per essere consultata sia dagli utenti sia dagli sviluppatori di Oracle, ma richiede una certa familiarità con i prodotti. La lettura delle prime pagine può essere d'aiuto per comprendere meglio la struttura della guida.

### 42.1 Contenuti della guida di riferimento

Questa guida di riferimento alfabetica comprende voci per qualsiasi comando di Oracle in SQL, PL/SQL e SQL/PLUS, oltre a definizioni per tutti i termini di una certa rilevanza utilizzati in Oracle e SQL. Ciascun comando viene riportato nella sintassi corretta e correlato di una spiegazione relativa allo scopo che si prefigge e all'utilizzo, al prodotto o ai prodotti in cui viene utilizzato, oltre a informazioni più dettagliate, limiti di utilizzo e suggerimenti, correlati da esempi per illustrarne l'impiego corretto. Gli argomenti sono in ordine alfabetico e presentano riferimenti incrociati, sia all'interno della guida di riferimento stessa, sia ai capitoli precedenti del libro.

### 42.2 Che cosa non contiene la guida di riferimento

Questa guida non è un'esercitazione su Oracle e non spiega il funzionamento degli strumenti di sviluppo con interfaccia grafica. Inoltre esistono alcune aree il cui utilizzo è talmente specializ-

zato e raro che si è ritenuto non necessario includerle in questa guida. In questi casi, il testo fa riferimento al manuale di Oracle, nel quale sarà possibile trovare le informazioni occorrenti.

### 42.3 Formato generale delle voci

Le voci di questa guida di riferimento sono generalmente definizioni di termini, oppure descrizioni di funzioni, comandi e parole chiave. Queste sono normalmente strutturate in sette parti: la parola chiave, il suo tipo, i prodotti in cui viene utilizzata, un riferimento incrociato “*Vedere anche*”, il formato in cui compare la parola chiave, una descrizione dei suoi componenti e un esempio di utilizzo. Ecco come appare una voce tipica:

#### **RPAD**

**VEDERE ANCHE** CARATTERI, FUNZIONI; LPAD, LTRIM, RTRIM, Capitolo 7

#### **SINTASSI**

RPAD(stringa, lunghezza [ , "set"])

**DESCRIZIONE** Abbreviazione di Right PAD (riempimento a destra). RPAD riempie una stringa fino a una certa lunghezza, aggiungendo alla sua destra una determinata sequenza di caratteri. Se non viene specificato il riempimento, viene aggiunto il carattere di riempimento predefinito: lo spazio.

#### **ESEMPIO**

select RPAD("HELLO ",24,"WORLD") from DUAL;

produce:

HELLO WORLDWORLDWORLDWOR

### **Parti che compongono ciascuna voce**

La PAROLA CHIAVE in genere compare su una riga a sé stante. In alcuni casi è seguita da una breve definizione. Se una parola chiave possiede più di una definizione, perché fa riferimento a strumenti di Oracle differenti, oppure a versioni diverse del programma, è seguita da una breve qualifica allo scopo di indicare che per quella parola chiave esiste più di una voce.

VEDERE ANCHE suggerisce altri argomenti strettamente correlati alla parola chiave, o i capitoli del libro che possono fornire descrizioni dettagliate sull'utilizzo pratico della stessa. Occasionalmente il lettore viene rimandato al manuale di Oracle o ad altri testi di riferimento che contengono dettagli che non rientrano nell'ambito di questa guida.

SINTASSI generalmente segue la notazione dei manuali di Oracle, in cui tutte le parole chiave, SQL e non, sono scritte in maiuscolo. In pratica, esse vanno scritte esattamente come vengono presentate (fatta eccezione per i casi in cui la distinzione tra maiuscole e minuscole non è importante). Le variabili e i parametri delle variabili sono scritti in minuscolo *corsivo*. Questo formato indica che il termine specificato dev'essere sostituito con il valore appropriato. Quando compaiono delle parentesi tonde, esse vanno scritte esattamente nel punto in cui appaiono, come se fossero parole scritte in maiuscolo.

## Standard per le variabili

Di seguito sono riportate alcune specifiche standard per le variabili.

VARIABILE	SIGNIFICATO
<i>Colonna</i>	Nome di una colonna.
<i>Database</i>	Nome di un database.
<i>Link</i>	Nome di un link in Oracle Net.
<i>Password</i>	Una password.
<i>Segmento</i>	Nome di un segmento.
<i>Tabella</i>	Nome di una tabella.
<i>Tablespace</i>	Nome di una tablespace.
<i>Utente</i>	Nome di un utente o proprietario.
<i>Vista</i>	Nome di una vista.

## Altre linee guida per la formattazione

Di seguito sono riportate altre linee guida riguardanti la formattazione.

- *carattere* significa che il valore dev'essere un unico carattere.
- *stringa* generalmente rappresenta una colonna di caratteri o un'espressione, o colonna, che può essere trattata come una colonna CHAR o VARCHAR2 dopo la conversione automatica dei dati.
- *valore* in genere rappresenta una colonna NUMBER o un'espressione, o colonna, che può essere trattata come una colonna NUMBER dopo la conversione automatica dei dati.
- *data* generalmente rappresenta una colonna di tipo DATA o un'espressione, o colonna, che può essere trattata come una colonna di tipo DATA dopo la conversione automatica dei dati.
- *intero* dev'essere un numero intero, come -3, 0 o 12.
- *espressione* può essere qualsiasi tipo di colonna. Il suo contenuto può essere un valore letterale, una variabile, un calcolo matematico, una funzione o una qualsiasi combinazione dei tipi appena enunciati; il risultato finale deve comunque essere un singolo valore, per esempio una stringa, un numero o una data.
- Occasionalmente, vengono utilizzate altre notazioni, come *condizione* o *query*. Ciò viene spiegato o risulta evidente nel contesto.
- Gli *elementi opzionali* vengono racchiusi tra parentesi quadre, come in [utente.], che significa che *utente* non è indispensabile.
- Gli *elementi alternativi* in un elenco vengono separati da un'unica barra verticale come per OFF | ON, che va letto "OFF oppure ON". Su alcuni sistemi questo simbolo viene visualizzato come una barra intera.
- *Opzioni richieste*; quando è richiesto un elemento di un elenco, esso viene racchiuso tra parentesi graffe, come {OFF | ON}.
- L'elemento *predefinito* in un elenco, se esiste, viene citato per primo.

- Tre puntini (o segno d’omissione) indicano che l’espressione precedente può essere ripetuta per un numero di volte qualsiasi, come in colonna [,colonna]...; in questo caso *colonna* rappresenta un numero qualsiasi di colonne aggiuntive, separate tra loro da virgolette.
- In qualche raro caso la notazione normale è insufficiente, o inappropriata, per ciò che ci si accinge a mostrare. In questi casi il testo del paragrafo “Descrizione” specifica in modo più completo quanto evidenziato in “Sintassi”.

## Altri elementi dell’elenco

Alcuni comandi hanno un *tipo restituito*, che indica il tipo di dato del valore restituito da una funzione.

**DESCRIZIONE** è una spiegazione verbale del comando e delle sue parti. Le parole in un tipo di carattere diverso contenute nella descrizione solitamente si riferiscono a riferimenti ad altri comandi o variabili mostrati nella parte “Sintassi”.

“Esempi” visualizza i risultati di una funzione, oppure come viene utilizzata una parola chiave in una query o in un comando reali. Lo stile degli esempi non è lo stesso adottato nella parte “Sintassi”. Al contrario, essi seguono lo stile delle prime quattro parti di questo libro (descritto nel Capitolo 3), poiché questo formato è tipico dell’utilizzo nella pratica normale.

### 42.4 Ordine delle parole chiave

Questa guida è in ordine alfabetico e le voci che iniziano con caratteri simbolici compaiono prima della prima voce che inizia con la lettera A.

Le parole con trattino, normale o di sottolineatura, compaiono nell’ordine come se il trattino normale o di sottolineatura fosse uno spazio.

## Simboli

I simboli vengono elencati in ordine di apparizione con una breve definizione o un nome. Quelli che hanno definizioni in **grassetto** possiedono un’intera voce a essi dedicata, oppure rappresentano prefissi a parole spiegate nelle pagine che seguono.

---

-	trattino di sottolineatura (detto anche underline, underscore o barra di sottolineatura).
!	punto esclamativo
"	doppi apici
#	cancelletto
\$	dollaro
?	punto interrogativo
%	segno di percentuale
&	“e” commerciale

---

(segue)

(continua)

&&	doppia "e" commerciale
'	apice singolo o apostrofo
()	parentesi tonde
*	asterisco o segno di moltiplicazione
**	esponenziale in PL/SQL
+	più
-	sottrazione o trattino
--	doppio trattino, segno meno o trattino nel commento SQL
.	punto, separatore di un nome o di una variabile
..	vai a
/	segno di divisione o barra
/*	barra asterisco, commento SQL diviso da o barra
:	due punti
:=	"uguale a" in PL/SQL
:	punto e virgola
<< >>	delimitatore di nome di etichetta in PL/SQL
<	minore di
<=	minore o uguale a
< >	diverso da
!=	diverso da
=	uguale a
>	maggiori di
>=	maggiori o uguali a
@	chiocciola
@@	doppia chiocciola
[]	parentesi quadre
^	accento circonflesso
^ =	diverso da
{}	parentesi graffe
	barra verticale spezzata
	concatenazione

## (trattino di sottolineatura)

Il trattino di sottolineatura rappresenta una singola posizione assieme all'operatore LIKE. Vedere LIKE.

### **\_EDITOR**

Vedere EDIT.

## **! (punto esclamativo)**

**VEDERE ANCHE** =, CONTAINS, SOUNDEX, Capitolo 24.

### **SINTASSI**

In SQL il punto esclamativo ! viene utilizzato come parte dell'espressione "diverso da" !=. Per esempio, di seguito vengono selezionate tutte le città che non appartengono al continente Asia:

```
select Citta
  from LOCALITA
 where Continente != "ASIA";
```

Per gli indici di testo CONTEXT, ! comunica al motore di ricerca dei testi di eseguire una ricerca SOUNDEX. I termini da ricercare vengono espansi in modo da includere termini che assomigliano al testo cercato, utilizzando il valore SOUNDEX del testo per stabilire le possibili corrispondenze; questo meccanismo vale quando si lavora con un testo in lingua inglese.

```
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, "!grate")>0;
```

## **" (doppi apici)**

**VEDERE ANCHE** ALIAS, TO\_CHAR, Capitolo 7 e Capitolo 9.

**DESCRIZIONE** " racchiude un alias di tabella o colonna che contiene caratteri speciali o uno spazio, oppure racchiude testi letterali in una clausola in formato data di TO\_CHAR.

### **ESEMPIO**

In questo esempio viene utilizzato come alias:

```
select GIORNO_SUCC(Ciclo_Data,"VENERDI") "Giorno di paga!"
  from GIORNOPAGA;
```

mentre di seguito viene utilizzato come una parte di formattazione di TO\_CHAR:

```
select Nome, Datanascita, TO_CHAR(Datanascita,
  '"Bambina nata il" ddth, fmMonth YYYY "alle" HH.MI "del mattino "")'
  "Formattata"
 from DATANASCITA
 where Nome = 'VITTORIA';
```

NOME	DATANASCITA	Formattata
VITTORIA	20-MAG-49	Bambina nata il 20 Mag 1949, alle 3.27 del mattino

## # (cancelletto)

**VEDERE ANCHE** DOCUMENT, REMARK, /\* \*/, Capitolo 6.

**DESCRIZIONE** # completa un blocco di documentazione in un file d'avvio di SQLPLUS, dove il blocco inizia con la parola DOCUMENT. SQL\*PLUS ignora tutte le linee a partire da quella con la parola DOCUMENT fino alla linea dopo il segno #.

## \$ (dollaro)

**VEDERE ANCHE** @, @@, HOST, START, CONTAINS, Capitolo 24.

### SINTASSI

Per SQL\*PLUS:

```
$ comando_host
```

Per gli indici di testo CONTEXT:

```
select colonna
      from tabella
     where CONTAINS(testo_colonna, '$termine_ricerca") >0;
```

**DESCRIZIONE** \$ trasferisce qualsiasi comando dell'host al sistema operativo per essere eseguito senza uscire da SQL\*PLUS. \$ è una forma abbreviata di HOST. Questa funzione purtroppo non garantisce il funzionamento su tutte le possibili piattaforme e sistemi operativi. Nel caso di un indice di testo CONTEXT, \$ istruisce il motore di ricerca in modo che esegua un'espansione etimologica (stem) del termine di ricerca. Un'espansione etimologica include quelle parole che hanno la stessa radice. Per esempio, un'espansione etimologica della parola 'work' include 'works', 'working' e così via.

## ? (punto interrogativo)

**VEDERE ANCHE** CONTAINS, Capitolo 24.

### SINTASSI

```
select colonna
      from tabella
     where CONTAINS(testo_colonna, '?termine") >0;
```

**DESCRIZIONE** Per gli indici di testo CONTEXT, ? segnala al motore di ricerca dei testi di eseguire una ricerca a corrispondenza non esatta. I termini da ricercare vengono espansi per includere parole che sono scritte in modo simile al termine cercato, utilizzando l'indice per i testi come fonte delle possibili corrispondenze.

## % (segno percentuale)

% è un carattere jolly utilizzato per rappresentare qualsiasi numero di posizioni e caratteri con l'operatore LIKE. Vedere LIKE.

## %FOUND

**VEDERE ANCHE** %ISOPEN, %NOTFOUND, %ROWCOUNT, CURSOR, SQL CURSOR, Capitolo 27.

### SINTASSI

*cursoro*%FOUND

oppure:

SQL%FOUND

**DESCRIZIONE** %FOUND rappresenta un flag di successo per select, insert, update e delete. *cursoro* è il nome di un cursore esplicito dichiarato in un blocco PL/SQL, oppure il cursore implicito di nome SQL. %FOUND può essere associato al nome di un cursore come suffisso. I due assieme rappresentano un flag di successo relativo all'esecuzione delle istruzioni select, insert, update e delete all'interno di blocchi PL/SQL.

PL/SQL impegna temporaneamente una parte di memoria come blocco degli appunti per l'esecuzione di istruzioni SQL e per memorizzare certi tipi di informazione (o *attributi*) relativi allo stato di questa esecuzione. Se l'istruzione SQL è select, quest'area contiene una riga di dati.

%FOUND rappresenta uno di questi attributi. %FOUND normalmente viene testato (con il ricorso alla logica IF/THEN) dopo che è stata eseguita un'istruzione fetch esplicita dal cursore. Sarà TRUE se l'istruzione fetch ottiene una riga, altrimenti sarà FALSE. Viene sempre valutato TRUE, FALSE o NULL. %NOTFOUND è il suo opposto logico. Risulta FALSE quando %FOUND è pari a TRUE, TRUE quando %FOUND è FALSE e NULL quando %FOUND è NULL.

L'esecuzione del controllo su %FOUND per conoscere la condizione di un cursore esplicito prima che venga aperto provoca un errore di tipo EXCEPTION (codice d'errore ORA-01001, INVALID CURSOR).

## %ISOPEN

**VEDERE ANCHE** CURSORE SQL, Capitolo 27.

### SINTASSI

*cursoro*%ISOPEN

**DESCRIZIONE** *cursoro* dev'essere il nome di un cursore dichiarato esplicitamente o il cursore implicito di nome SQL. Viene valutato TRUE se il cursore specificato risulta aperto, FALSE se non lo è. SQL%ISOPEN viene sempre valutato pari a FALSE, perché il cursore SQL viene aperto e chiuso automaticamente quando viene eseguita un'istruzione SQL non esplicitamente dichiarata (*vedere CURSORE SQL*). %ISOPEN viene utilizzato nella logica PL/SQL; non può far parte di un'istruzione SQL.

## %NOTFOUND

*Vedere* %FOUND.

## %ROWCOUNT

**VEDERE ANCHE** CLOSE, DECLARE, DELETE, FETCH, INSERT, OPEN, SELECT, UPDATE, Capitolo 27.

### SINTASSI

*cursoro*%ROWCOUNT

**DESCRIZIONE** *cursoro* rappresenta il nome di un cursore dichiarato in modo esplicito, oppure il cursore隐式 di nome SQL. *cursoro*%ROWCOUNT contiene il numero totale cumulativo di righe restituite tramite FETCH dall'insieme attivo nel cursore attuale. Può essere utilizzato per elaborare intenzionalmente solo un numero prefissato di righe, ma più comunemente viene utilizzato come un gestore di eccezioni per le operazioni select predisposte per restituire una sola riga (per esempio select...into). In questi casi, se non viene restituita alcuna riga, %ROWCOUNT viene impostato a 0 (questo controllo può essere effettuato anche tramite %NOTFOUND; nel caso in cui venga restituita più di una riga, non importa quante, il suo valore viene impostato a 2).

%ROWCOUNT è utilizzato nella logica PL/SQL; non può entrare a far parte di un'istruzione SQL. Se viene utilizzato SQL%ROWCOUNT, esso può fare riferimento solo al cursore implicito aperto più recente. Se nessun cursore implicito è stato aperto, SQL%ROWCOUNT restituisce NULL.

## %ROWTYPE

**VEDERE ANCHE** FETCH, Capitolo 27.

### SINTASSI

{[*utente*.]*tabella* | *cursoro*}%ROWTYPE

**DESCRIZIONE** %ROWTYPE dichiara una variabile di un record in modo che abbia la stessa struttura di un'intera riga di una tabella (o vista), oppure di una riga recuperata tramite il cursore specificato. %ROWTYPE viene utilizzato come parte della dichiarazione di una variabile e fa in modo che la variabile contenga i campi appropriati e i tipi di dati per gestire tutte le colonne riportate. Se viene specificato [*utente*].*tabella*, la tabella (o vista) specificata deve esistere nel database.

Per creare una variabile che contenga i campi corrispondenti, occorre utilizzare declare, il nome di una variabile e %ROWTYPE con il nome della tabella, per poi selezionare una riga nel record. Dato che la variabile ha la stessa struttura della tabella, è possibile fare riferimento ai campi come *variabile.campo*, utilizzando una notazione simile a quella *tabella.colonna* propria delle istruzioni SQL. select...into e fetch...into rappresentano i soli metodi per caricare un intero record in una variabile. I singoli campi all'interno del record possono essere caricati utilizzando la notazione :=, come mostrato di seguito:

```
BIBLIOTECA_RECORD.Editore := "PANDORA";
```

Se come prefisso per %ROWTYPE si utilizza un cursore, esso può contenere un'istruzione select con tante colonne quante sono necessarie. Tuttavia, se una colonna prelevata da un cursore specifico è un'espressione, anziché un semplice nome di colonna, a questa espressione dev'essere dato un alias nell'istruzione select prima che sia possibile farvi riferimento tramite questo metodo.

## %TYPE

**VEDERE ANCHE** %ROWTYPE, Capitolo 27.

### SINTASSI

```
{[utente.]tabella.colonna | variabile}%TYPE
```

**DESCRIZIONE** %TYPE viene utilizzato per dichiarare una variabile dello stesso tipo di quella dichiarata precedentemente, oppure come colonna particolare di una tabella già esistente nel database cui si è collegati.

### ESEMPIO

In questo esempio viene dichiarata una nuova variabile, Scrittore, in modo che sia dello stesso tipo della colonna NomeAutore della tabella AUTORE. Dal momento che Scrittore ora esiste, può essere utilizzata per dichiarare un'altra nuova variabile, Coautore.

```
Scrittore  AUTORE.NomeAutore%TYPE;
Coautore  Scrittore%TYPE;
```

## & o && (e commerciale o e commerciale doppia)

**VEDERE ANCHE** :, ACCEPT, DEFINE, START, CONTAINS, Capitolo 14 e Capitolo 24.

### SINTASSI

Per SQL\*PLUS:

```
&intero
&variabile
&&variabile
```

Per gli indici CONTEXT:

```
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, "proprietà e raccolti")>0;
```

**DESCRIZIONE** & e && possono essere utilizzati in diversi modi (&& si applica solamente alla seconda definizione descritta di seguito).

- Prefisso dei parametri di un file di avvio di SQL\*PLUS. &1, &2 e così via vengono sostituiti da valori. Vedere START.
- Prefisso di una variabile di sostituzione di un comando SQL in SQL\*PLUS. SQL\*PLUS richiede l'inserimento di un valore se trova una variabile & o && non definita. && definisce la variabile e quindi ne preserva il valore; & non definisce o preserva il valore, ma semplicemente esegue una sostituzione di ciò che viene inserito. Vedere ACCEPT e DEFINE.
- Quando si impiegano indici CONTEXT, & può inoltre essere utilizzato per segnalare una condizione AND per ricerche di testi che coinvolgono più termini. Se anche solo uno dei termini ricercati non viene trovato, il testo non viene restituito dall'operazione di ricerca.

## ' (apice singolo)

**VEDERE ANCHE** " (doppi apici), Capitolo 7.

**SINTASSI**

"stringa"

**DESCRIZIONE** Gli apici singoli racchiudono dei letterali (come stringhe di caratteri o date). Per specificare un apice o un apostrofo all'interno di una costante di tipo stringa, è necessario inserire due simboli ' (non un doppio apice). È opportuno evitare l'utilizzo di apostrofo nelle date (e altrove) ogni volta che sia possibile.

**ESEMPIO**

select "Guglielmo" from DUAL;

produce:

Guglielmo

mentre l'istruzione seguente:

select "Guglielmo l""attore" from DUAL;

produce:

Guglielmo l"attore

**( ) (parentesi)**

**VEDERE ANCHE** PRECEDENZA, SUBQUERY, UPDATE, Capitoli 4, 12 e 15.

**DESCRIZIONE** Delimita subquery, elenchi di colonne o precedenze di controlli all'interno delle espressioni.

**\* (moltiplicazione)**

**VEDERE ANCHE** +, -, /, Capitoli 8 e 24.

**SINTASSI**

*valore1* \* *valore2*

**DESCRIZIONE** *valore1* \* *valore2* significa *valore1* moltiplicato per *valore2*. Per gli indici CONTEXT, \* viene utilizzato per ponderare in modo diverso i singoli termini ricercati.

**\*\* (esponenziale)**

**VEDERE ANCHE** POWER.

**SINTASSI**

*x* \*\* *y*

**DESCRIZIONE** Il valore *x* viene elevato alla potenza *y*. *x* e *y* possono essere costanti, variabili, colonne o espressioni. Entrambe devono essere di tipo numerico.

**ESEMPIO**

$4^{**}4 = 256$

**+ (addizione)**

**VEDERE ANCHE** -, \*, /, Capitolo 8.

**SINTASSI**

*valore1* + *valore2*

**DESCRIZIONE** *valore1* + *valore2* significa *valore1* più *valore2*.

**- (sottrazione [forma 1])**

**VEDERE ANCHE** +, \*, /, Capitolo 8, CONTAINS, MINUS, Capitolo 24

**SINTASSI**

*valore1* - *valore2*

**DESCRIZIONE** *valore1* – *valore2* significa *valore1* meno *valore2*. Per gli indici CONTEXT, un segno meno ‘–’ indica la ricerca, all’interno del testo, di due termini per sottrarre il punteggio ottenuto dalla ricerca del secondo termine da quello della ricerca del primo, prima di confrontare il risultato con il punteggio di soglia. Per gli indici CTXCAT, un segno meno ‘–’ segnala al motore di ricerca di escludere la riga dal set di risultati se il termine che segue il segno ‘–’ viene trovato.

**- (trattino [forma 2])**

**VEDERE ANCHE** Capitolo 14.

**SINTASSI**

*testo del comando* -  
    *testo* -  
    *testo*

**DESCRIZIONE** Continuazione di un comando SQL\*PLUS. - continua un comando sulla linea seguente.

**ESEMPIO**

```
ttitle left 'Portafoglio attuale' -  
        right '1 Aprile 1998'      skip 1 -  
        center 'Elenchi Industrie' skip 4;
```

**-- (commento)**

**VEDERE ANCHE** /\* \*/, REMARK, Capitolo 6.

**SINTASSI**

```
-- qualsiasi testo
```

**DESCRIZIONE** -- comunica a Oracle l'inizio di un commento. Qualsiasi cosa sia presente da questo punto fino alla fine della linea viene trattato come un commento. Questi delimitatori vengono utilizzati solo all'interno di SQL stesso o in PL/SQL e devono apparire prima del terminatore SQL.

**ESEMPIO**

```
select Numero, Rubrica, Pagina
-- questo è un commento
from GIORNALE    -- questo è un altro commento
where Rubrica = "F"
```

**. (punto [forma 1])****SINTASSI**

```
&variabile.suffisso
```

**DESCRIZIONE** Il punto . rappresenta un separatore di variabile, utilizzato in SQL\*PLUS per separare il nome della variabile da un suffisso in modo che quest'ultimo non venga considerato come parte del nome della variabile.

**ESEMPIO**

Qui il suffisso “a” è effettivamente concatenato al contesto della variabile &Viale.

```
define Viale = 21
select '&Viale.a Strada n 100' from DUAL;
```

produce:

21a Strada n. 100

La stessa tecnica può anche essere utilizzata in una clausola where.

**. (punto [forma 2])**

**VEDERE ANCHE** OPERATORI SINTATTICI, Capitoli 4 e 21.

**SINTASSI**

```
[utente.][tabella.]colonna
```

**DESCRIZIONE** Il punto . è un separatore di nomi, utilizzato per specificare il nome completo di una colonna, includendo (facoltativamente) la relativa tabella o l'utente. Viene inoltre utilizzato per specificare colonne all'interno di tipi di dati astratti nelle operazioni select, update e delete.

**ESEMPIO**

```
select Pratica.BIBLIOTECA.Titolo
      from Pratica.BIBLIOTECA, Pratica.BIBLIOTECA_AUTORE
```

```
where Pratica.BIBLIOTECA.Titolo = Pratica.BIBLIOTECA_AUTORE.Titolo;
```

Se la tabella contiene colonne basate su un tipo di dato astratto, l'accesso agli attributi di quest'ultimo può avvenire con il ricorso alla notazione . (punto).

```
select C.Persona.Indirizzo.Citta  
  from Societa C  
 where C.Persona.Indirizzo.Provincia = "MI";
```

*Vedere il Capitolo 4 per ulteriori esempi.*

## .. (a)

*Vedere CICLI.*

### / (divisione [forma 1])

**VEDERE ANCHE** +, -, \*, Capitolo 8.

#### SINTASSI

*valore1 / valore2*

**DESCRIZIONE** *valore1 / valore2* significa *valore1* diviso per *valore2*.

### / (barra [forma 2])

**VEDERE ANCHE** ;, BUFFER, EDIT, GET, RUN, SET, SQLTERMINATOR, Capitolo 14

**DESCRIZIONE** / esegue l'istruzione SQL nel buffer SQL senza visualizzarla, contrariamente a RUN, che prima di eseguirla la visualizza.

### /\* \*/ (commento)

**VEDERE ANCHE** REMARK, Capitolo 6.

#### SINTASSI

*/\* testo qualsiasi \*/*

**DESCRIZIONE** /\* comunica a Oracle l'inizio di un commento. Qualunque cosa che Oracle incontri da quel punto in avanti, anche molte parole su più righe, viene considerato un commento, fino a che non si incontra il delimitatore di fine commento \*/. Non è possibile racchiudere commenti all'interno di altri commenti; in altre parole, un /\* viene terminato dal primo \*/ che si incontra, anche se tra i due è presente un altro delimitatore /\* di apertura di commento.

#### ESEMPIO

```
select Argomento, Sezione, Pagina  
/* questo è un commento su più righe  
   usato per documentare ampiamente il codice */  
from GIORNALE  
where Sezione = "F";
```

**: (due punti, prefisso di variabile host)****VEDERE ANCHE** VARIABILE INDICATORE.**SINTASSI***:nome*

**DESCRIZIONE** *nome* è il nome di una variabile host. Quando si incorpora PL/SQL in un linguaggio host attraverso un precompilatore di Oracle, è possibile fare riferimento alle variabili host dai blocchi PL/SQL, facendo precedere i loro nomi nel linguaggio host con un segno di due punti. A tutti gli effetti si fa riferimento alla variabile host. Se PL/SQL cambia il suo valore attraverso un assegnamento (*:=*), il valore della variabile host viene modificato. Esse possono essere utilizzate ovunque possa essere specificata una variabile PL/SQL. L'unica eccezione consiste nell'assegnamento del valore NULL a una variabile host, che non viene supportato direttamente, ma richiede l'utilizzo di una variabile indicatore.

**ESEMPIO**

```
int ConteggioLibri;
...
select CONTEGGIO(*) into :ConteggioLibri from BIBLIOTECA;
```

**:= (uguale a)****VEDERE ANCHE** DECLARE, FETCH, SELECT INTO, Capitolo 27.**SINTASSI***variabile* := *espressione*

**DESCRIZIONE** La *variabile* PL/SQL viene impostata uguale a *espressione*, che può essere una costante, NULL oppure un'operazione di calcolo con altre variabili, lettere e funzioni di PL/SQL.

**ESEMPIO**

```
Pro := Quantita * Prezzo;
Titolo := "BARBIERI SBARBATI";
Nome := NULL
```

**; (punto e virgola)****VEDERE ANCHE** / (barra), BUFFER, EDIT, GET, RUN, SQLTERMINATOR, CONTAINS, NEAR, Capitolo 24.

**DESCRIZIONE** ; esegue l'istruzione SQL o il comando che lo precede. Per gli indici CONTEXT indica di effettuare una ricerca di vicinanza per le stringhe di testo specificate. Se i termini da cercare fossero 'estate' e 'vacanza', una ricerca di prossimità potrebbe essere strutturata nella maniera seguente:

```
select Testo
  from SONNET
 where CONTAINS(Testo,"estate;vacanza") >0;
```

Quando si valutano i risultati della ricerca testuale, le righe contenenti 'estate' e 'vacanza' in stretta prossimità ottengono punteggi maggiori di quelle in cui le due parole distano di più.

## <<> (delimitatore di nome dell'etichetta PL/SQL)

Vedere BEGIN; BLOCCO, STRUTTURA; END; GOTO; CICLI; Capitolo 27.

### @ (chiocciola [forma 1])

**VEDERE ANCHE** @@, START.

#### SINTASSI

`@file`

**DESCRIZIONE** @ esegue il file d'avvio di SQL\*PLUS di nome *file*. @ è simile a START, ma non accetta argomenti sulla linea di comando.

### @ (chiocciola [forma 2])

**VEDERE ANCHE** CONNECT, COPY, LINK DI DATABASE, Capitolo 22.

#### SINTASSI

```
CONNECT [utente[/password] [@database];
        COPY [FROM utente/password@database]
              [TO utente/password@database]
              {APPEND | CREATE | INSERT | REPLACE}
              table [ (colonna, [colonna]...) ]
              USING query;
        SELECT...    FROM [utente.]tabella[link] [, [utente.]tabella[link]] ...
```

**DESCRIZIONE** @ rappresenta il prefisso al nome del database in un comando CONNECT, COPY o nel nome di un collegamento all'interno di una clausola from.

### @@ (doppia chiocciola)

**VEDERE ANCHE** @ (Forma 1), START.

#### SINTASSI

`@@file`

**DESCRIZIONE** @@ esegue un file d'avvio di SQL\*PLUS annidato, di nome *file*. @@ è simile a @, ma differisce da esso in quanto, se utilizzato in un file di comandi, ricerca il file d'avvio nella stessa directory del file di comandi che lo richiama (anziché nella directory in cui ci si trova quando si esegue il file di comandi).

### { } (parentesi graffe)

**VEDERE ANCHE** CONTAINS, Capitolo 24, Capitolo 34.

#### SINTASSI

Per le ricerche di testo, {} indica che il testo racchiuso dev'essere considerato parte di una stringa di ricerca anche se si tratta di una parola riservata. Per esempio, se il termine ricercato è 'in and out' e si desidera ritrovare l'intera frase, è necessario racchiudere la parola 'and' tra parentesi graffe:

```
REM Metodo CONTAINS per gli indici CONTEXT:
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, transazioni {e} finanze")>0;
```

In Java le parentesi graffe {} indicano l'inizio e la fine di un blocco. Vedere il Capitolo 34 per degli esempi di blocchi Java.

## | (barra verticale )

**VEDERE ANCHE** BTITLE, HEADSEP, TTITLE, Capitolo 6, Capitolo 24.

### SINTASSI

*testo|testo*

**DESCRIZIONE** Quando viene utilizzato in column, ttitle o btitle di SQL\*PLUS, | è il carattere headsep predefinito e serve a indicare la suddivisione di una riga in una seconda riga (quando viene utilizzato negli elenchi di questa guida di riferimento, rappresenta il carattere separatore tra scelte alternative: variabile | lettera va letto “variabile o lettera”). Nelle query sul testo, | segnala una condizione OR per le ricerche che coinvolgono più termini di ricerca. Nel caso in cui uno dei termini cercati venga trovato e il suo punteggio superi il valore di soglia specificato, il testo viene restituito dalla ricerca.

### ESEMPIO

Di seguito viene riportato l'utilizzo di | come carattere headsep.

TTITLE 'Questa è la prima riga|e questa è la seconda"

produce:

Questa è la prima riga  
e questa è la seconda

Ecco invece come viene utilizzato come clausola OR in una query di ricerca dei testi:

```
REM Metodo CONTAINS per gli indici CONTEXT:
select Titolo
  from CONTEXT_RECENSIONI_LIBRI
 where CONTAINS(Testo_Recensione, "proprietà | raccolti")>0;
REM Metodo CATSEARCH per gli indici CTXCAT:
select Titolo
  from CTXCAT_RECENSIONI_LIBRI
 where CATSEARCH(Testo_Recensione, "proprietà | raccolti", NULL)>0;
```

## || (concatenazione)

**VEDERE ANCHE** . (punto [forma 1]), CONCAT, SUBSTR, Capitolo 7.

### SINTASSI

*espressione1 || espressione2*

**DESCRIZIONE** || concatena due stringhe.

**ESEMPIO**

Si utilizza || per visualizzare una colonna di città, seguite da una virgola, uno spazio e dal paese a cui appartengono:

```
select Citta||", "||Paese from LOCALITA
```

**ABS (valore assoluto)**

**VEDERE ANCHE** NUMERI, FUNZIONI; Capitolo 8.

**SINTASSI**

```
ABS(valore)
```

*valore* dev'essere un numero, sia un numero di tipo letterale, sia il nome di una colonna numerica, oppure una stringa di caratteri contenente un numero valido o ancora una colonna di caratteri contenente solo numeri validi.

**DESCRIZIONE** Il valore assoluto è la misura della grandezza di un elemento ed è sempre un numero positivo.

**ESEMPI**

```
ABS(146) = 146  
ABS(-30) = 30  
ABS("-27.88") = 27.88
```

**ACCEPT**

**VEDERE ANCHE** &, &&, DEFINE, Capitolo 14.

**SINTASSI**

```
ACC[EPT] variabile [NUM[BER]|CHAR|DATE] [FOR[MAT]] formato  
[DEF[AULT] default] [PROMPT testo | NOPR[OMPT]] [HIDE]
```

**DESCRIZIONE** ACCEPT preleva l'input dalla tastiera dell'utente e lo registra nella variabile specificata. Se la variabile non è stata prima definita tramite DEFINED, viene creata. NUMBER, CHAR o DATE determinano il tipo di dato della variabile nel momento in cui viene inserita. CHAR accetta qualsiasi numero o carattere. DATE accetta stringhe di caratteri con formati di data validi (in caso contrario, viene restituito un messaggio d'errore). NUMBER accetta solo numeri, con un punto decimale e un segno meno opzionali, altrimenti ACCEPT produce un messaggio d'errore. FORMAT specifica il formato di input per la risposta (come A20). DEFAULT imposta il valore predefinito, nel caso non venga data una risposta. PROMPT visualizza il testo all'utente prima di accettare l'input. NOPROMPT salta una riga e aspetta l'input senza visualizzare alcun prompt. Se non si specifica né PROMPT né NOPROMPT, ACCEPT inventa un prompt per richiedere di introdurre il nome per la variabile. HIDE occulta il valore inserito dall'utente ed è utile per password e simili.

**ESEMPIO**

L'esempio che segue chiede all'utente "Che temperatura c'è?" e registra il dato inserito dall'utente in una variabile di nome Temperatura:

```
ACCEPT Temperatura NUMBER PROMPT "Che temperatura c'è? "
```

## ACCESSO CONCORRENTE

*Accesso concorrente* è un termine generale che indica l'accesso allo stesso dato da parte di più utenti. Nel software per database, l'accesso concorrente richiede una programmazione software complessa per assicurare che tutti gli utenti abbiano accesso ai dati corretti e che ogni modifica venga effettuata nell'ordine appropriato.

## ACCUMULATE

*Vedere* CONTAINS.

## ACOS

**VEDERE ANCHE** ASIN, ATAN, ATAN2, COS, COSH, EXP, LN, LOG, SIN, SINH, TAN, TANH.

### SINTASSI

ACOS(*valore*)

**DESCRIZIONE** ACOS restituisce l'arcocoseno di un valore. I valori di input variano da -1 a 1; gli output sono espressi in radianti.

## ADD\_MONTHS

**VEDERE ANCHE** DATE, FUNZIONI; Capitolo 9.

### SINTASSI

ADD\_MONTHS(*data, intero*)

**DESCRIZIONE** ADD\_MONTHS aggiunge un numero di mesi a *data* e restituisce la data come sarà, in futuro, dopo quei mesi. *data* dev'essere una data di Oracle valida. *intero* rappresenta un valore intero; un valore non intero viene troncato al più piccolo intero successivo. Se *intero* ha valore negativo, viene restituita una data del passato.

### ESEMPIO

ADD\_MONTHS aggiunge 1 mese e 0,3 mesi alla data del 22 aprile 2001:

```
select ADD_MONTHS(TO_DATE("22-APR-01"),1),
       ADD_MONTHS(TO_DATE("22-APR-01"),.3) from DUAL;
-----
ADD_MONTH ADD_MONTH
-----
22-MAY-01 22-APR-01
```

## ALIAS

L'alias rappresenta un nome temporaneo assegnato a una tabella o ad una colonna all'interno di un'istruzione SQL e viene utilizzato per fare riferimento a questi elementi in un altro punto nella stessa istruzione (se si tratta di una tabella) o in un comando SQL\*PLUS (se si tratta di una

colonna). Per separare la definizione della colonna dal proprio alias, è possibile utilizzare la parola chiave AS. Vedere " (apici doppi), AS e SELECT. Quando si ricorre a un alias per una tabella, esso viene chiamato *nome di correlazione*.

## ALIAS DI TABELLA

L'alias di tabella è un sostituto temporaneo per un nome di tabella. Viene definito nella clausola from dell'istruzione select. Vedere AS e il Capitolo 11.

## ALL

**VEDERE ANCHE** ANY, BETWEEN, EXISTS, IN, OPERATORI LOGICI, Capitolo 12.

### SINTASSI

*operatore ALL elenco*

**DESCRIZIONE** != ALL è l'equivalente di NOT IN. *operatore* può essere =, >, >=, <, <=, !=, mentre *elenco* rappresenta una serie di stringhe letterali (quali 'Mario', 'Giorgio', 'Ilaria') oppure una serie di numeri letterali (come 2, 43, 76, 32.06, 444) o una colonna da una query secondaria, dove ciascuna riga della subquery diventa un membro dell'elenco, come per:

LOCALITA.Citta != ALL (select Citta from CLIMA)

Può inoltre rappresentare una serie di colonne nella clausola where della query principale, come mostrato di seguito:

Prospetto != ALL (Venditore, Cliente)

### LIMITAZIONI

*elenco* non può rappresentare una serie di colonne all'interno di una subquery, come:

Prospetto != ALL (select Venditore, Cliente from . . .)

Molti trovano che questo operatore e l'operatore ANY siano alquanto difficili da memorizzare, perché la logica utilizzata da entrambi in alcuni casi non è immediatamente intuitiva. Di conseguenza vengono solitamente sostituiti con alcune forme di EXISTS. La combinazione di un operatore con ALL e un elenco può risultare più chiara dalla seguente tabella.

Pag = ALL (4,2,7)	Pag è uguale a ciascun elemento dell'elenco. Nessun numero soddisfa la condizione. Se una subquery restituisse un elenco in cui tutti gli elementi siano identici, il valore di Pag potrebbe essere uguale a ciascuno di essi, ma si tratta di un caso raro.
Pag > ALL (4,2,7)	Pag è maggiore dell'elemento più grande nell'elenco (4,2,7). Qualsiasi elemento maggiore di 7 soddisfa la condizione.
Pag >= ALL (4,2,7)	Pag è maggiore o uguale all'elemento più grande nell'elenco (4,2,7). Qualsiasi elemento maggiore o uguale a 7 soddisfa la condizione.
Pag < ALL (4,2,7)	Pag è minore del più piccolo elemento nell'elenco (4,2,7). Qualsiasi elemento minore di 2 soddisfa la condizione.

(segue)

(continua)

---

Pag <= ALL (4,2,7) Pag è minore o uguale all'elemento più piccolo nell'elenco (4,2,7). Qualsiasi elemento minore o uguale a 2 soddisfa la condizione.

---

Pag != ALL (4,2,7) Pag è diverso da qualsiasi elemento dell'elenco. Qualsiasi numero soddisfa la condizione, fatta eccezione per 4, 2 e 7.

---

## ALLOCATE (interno SQL)

**VEDERE ANCHE** DECLARE CURSOR, documentazione del precompilatore.

### SINTASSI

```
EXEC SQL [AT {db_nome | :variabile_host} ]
  ALLOCATE { :variabile_cursore | :ptr_host }
  [[INDICATOR] :ptr_ind]
```

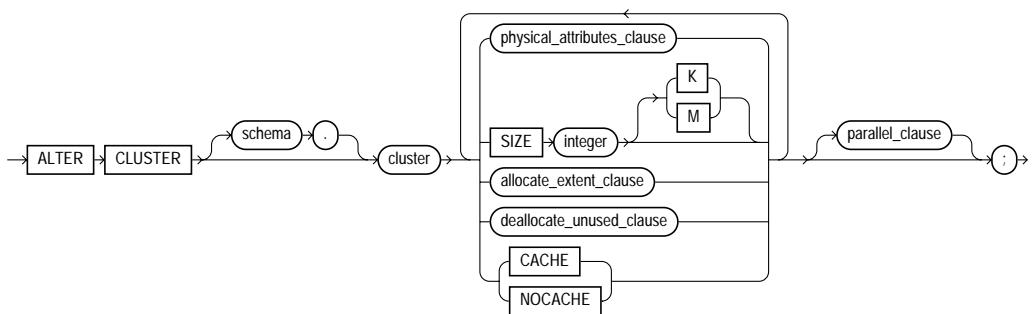
**DESCRIZIONE** La clausola ALLOCATE alloca una variabile cursore cui sia possibile fare riferimento in un blocco PL/SQL. Consultare la documentazione del precompilatore per maggiori dettagli su ALLOCATE e le clausole correlate, come il puntatore host.

## ALTER CLUSTER

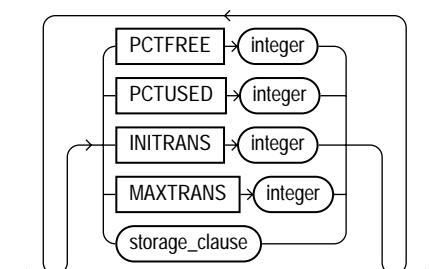
**VEDERE ANCHE** CREATE CLUSTER, CREATE TABLE, STORAGE, Capitolo 20.

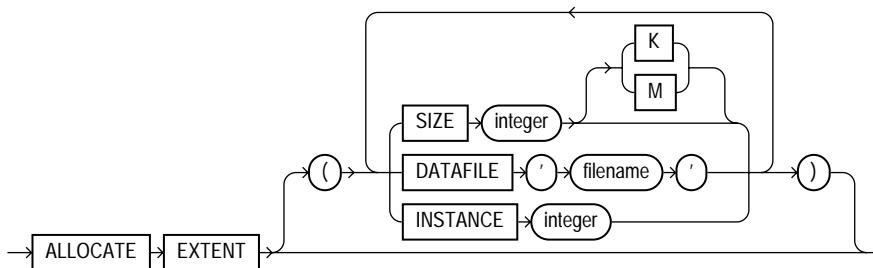
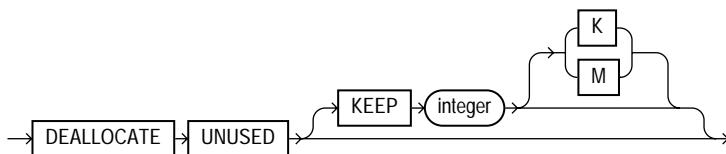
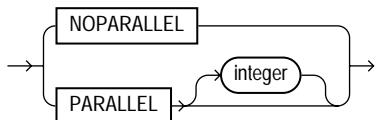
### SINTASSI

alter\_cluster



### physical\_attributes\_clause



**allocate\_extent\_clause****deallocate\_unused\_clause****parallel\_clause**

**DESCRIZIONE** Le descrizioni dei parametri si trovano alla voce CREATE TABLE. I parametri di ALTER CLUSTER hanno lo stesso scopo ed effetto di quelli CREATE TABLE, tranne per il fatto che vengono utilizzati per modificare un cluster. SIZE viene descritto alla voce CREATE CLUSTER. Per modificare un cluster, è necessario esserne il proprietario o disporre del privilegio di sistema ALTER ANY CLUSTER.

Tutte le tabelle di un cluster utilizzano i valori per i parametri PCTFREE, PCTUSED, INITTRANS, MAXTRANS, TABLESPACE e STORAGE impostati da CREATE CLUSTER. ALTER CLUSTER modifica questi valori per i blocchi di cluster successivi, ma non ha effetto su quelli già esistenti. ALTER CLUSTER non consente di modificare il parametro MINEXTENTS in STORAGE. La clausola DEALLOCATE UNUSED consente al cluster di compattare la propria dimensione, liberando dello spazio inutilizzato.

**ESEMPIO**

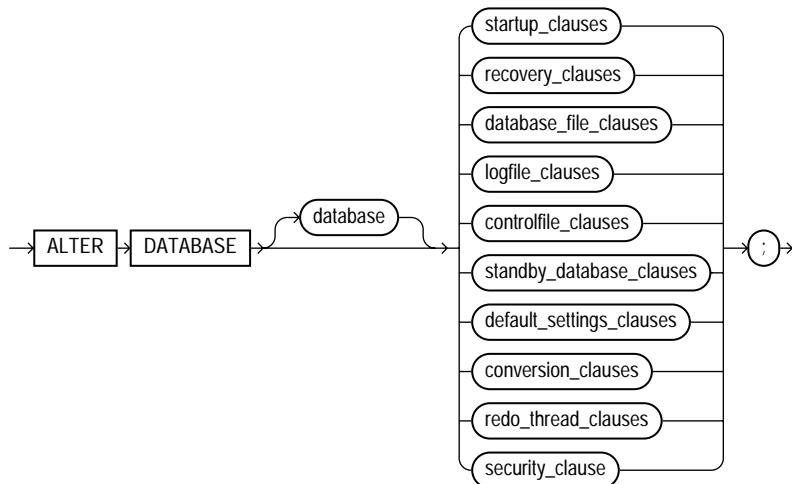
```
alter cluster LIBROandAUTORE size 1024;
```

**ALTER DATABASE**

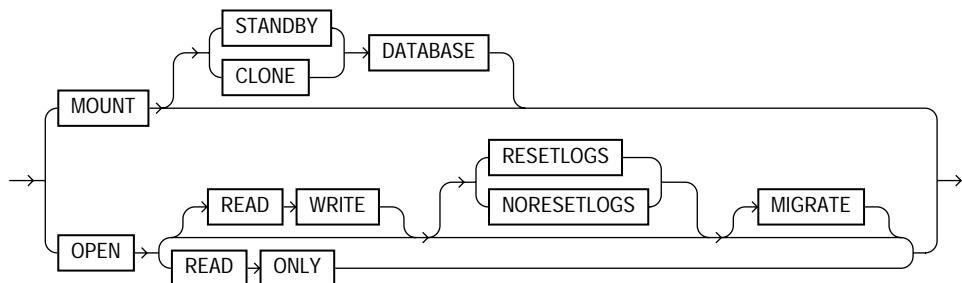
**VEDERE ANCHE** ALTER ROLLBACK SEGMENT, AVVIO, CREATE DATABASE, RECOVER, SHUTDOWN, Capitolo 40.

## SINTASSI

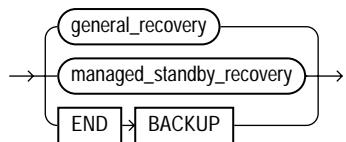
`alter_database`

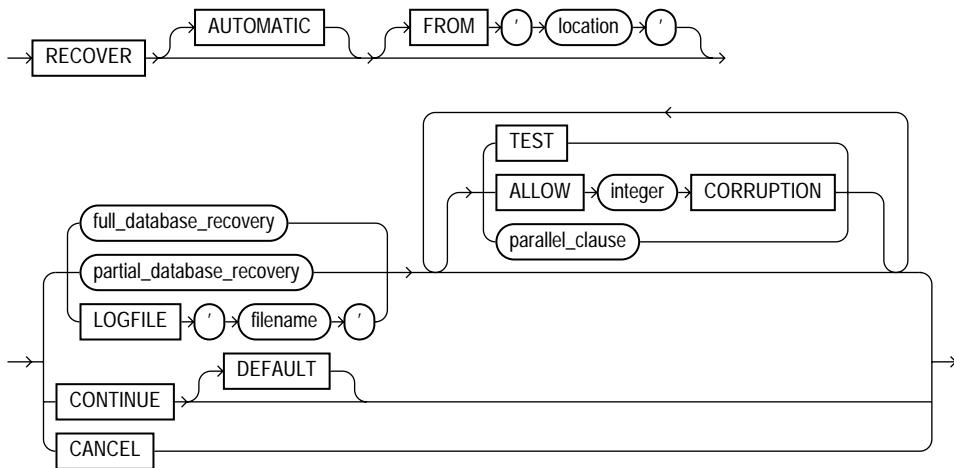
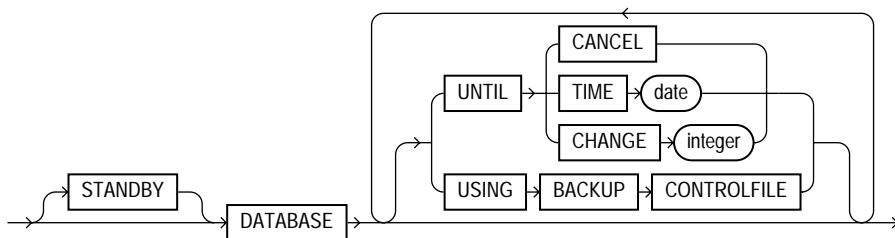
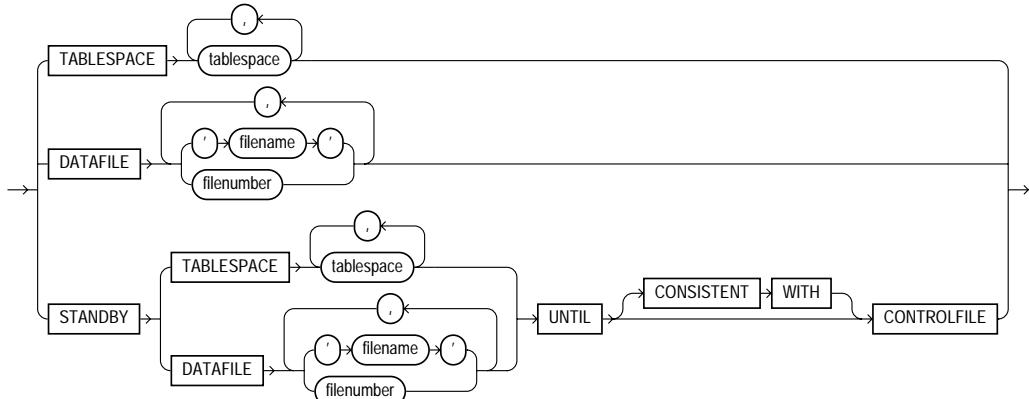
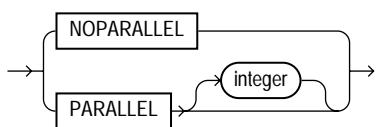


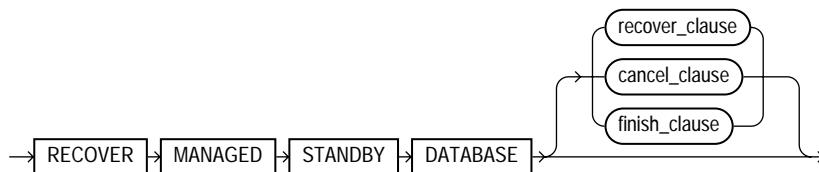
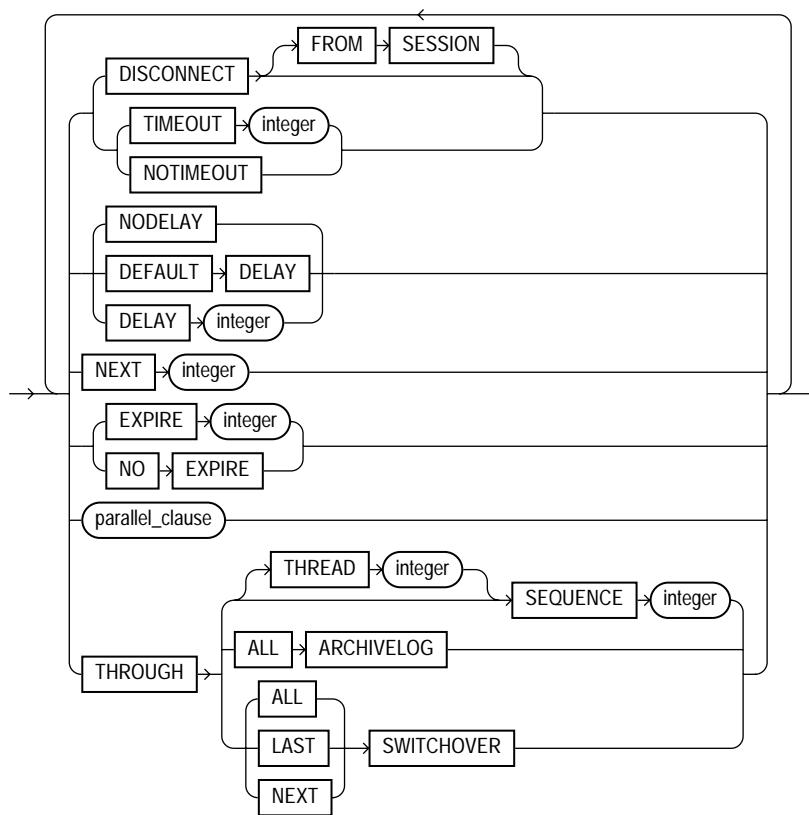
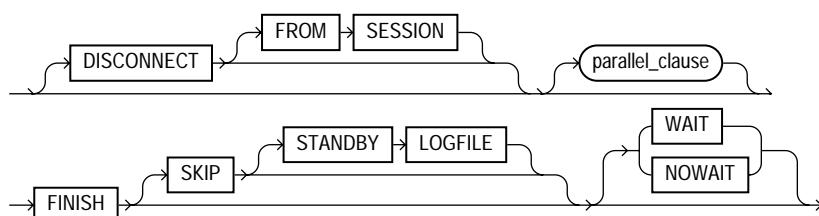
`startup_clauses`

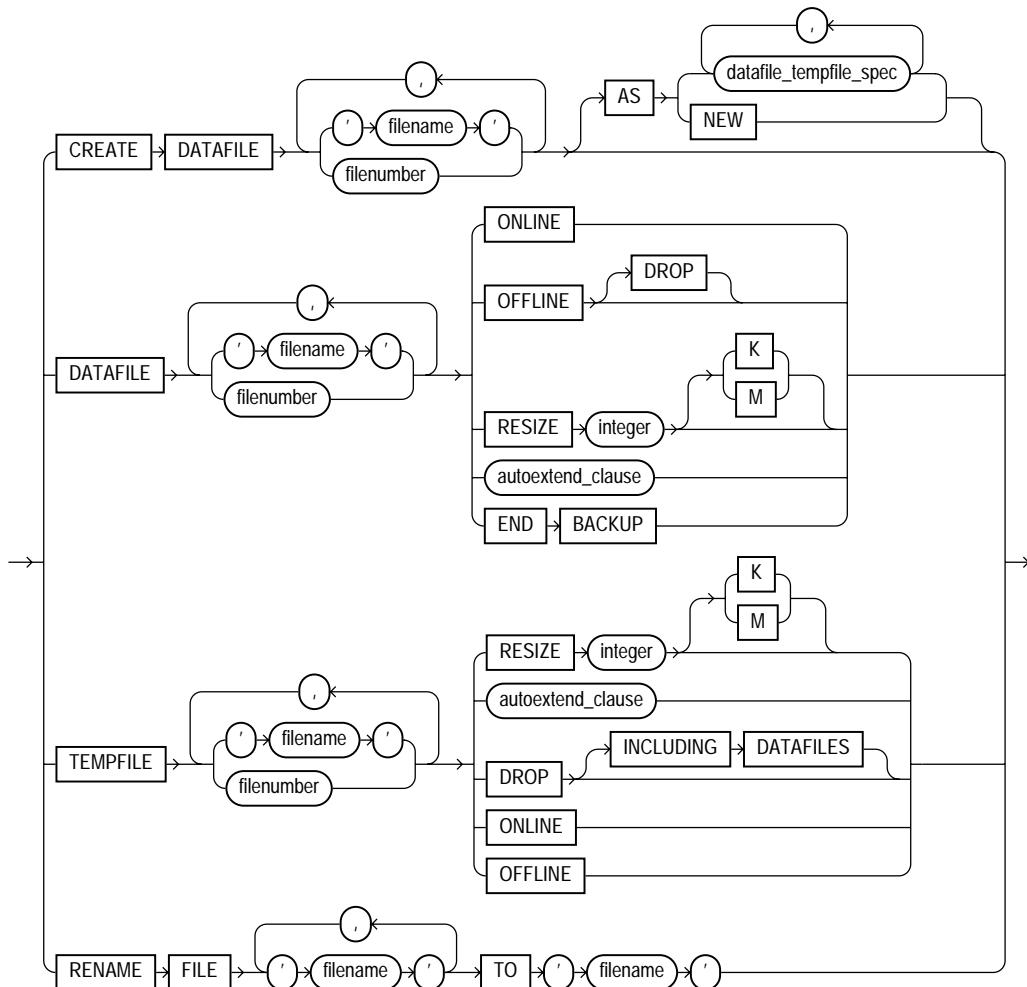
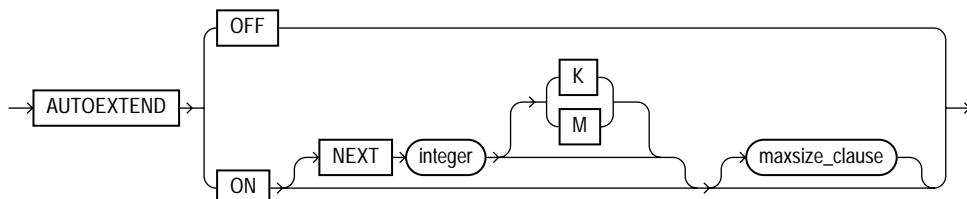
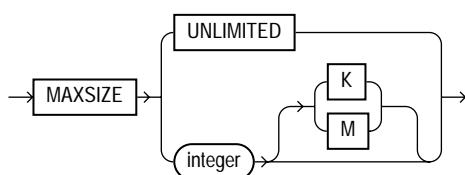


`recovery_clauses`

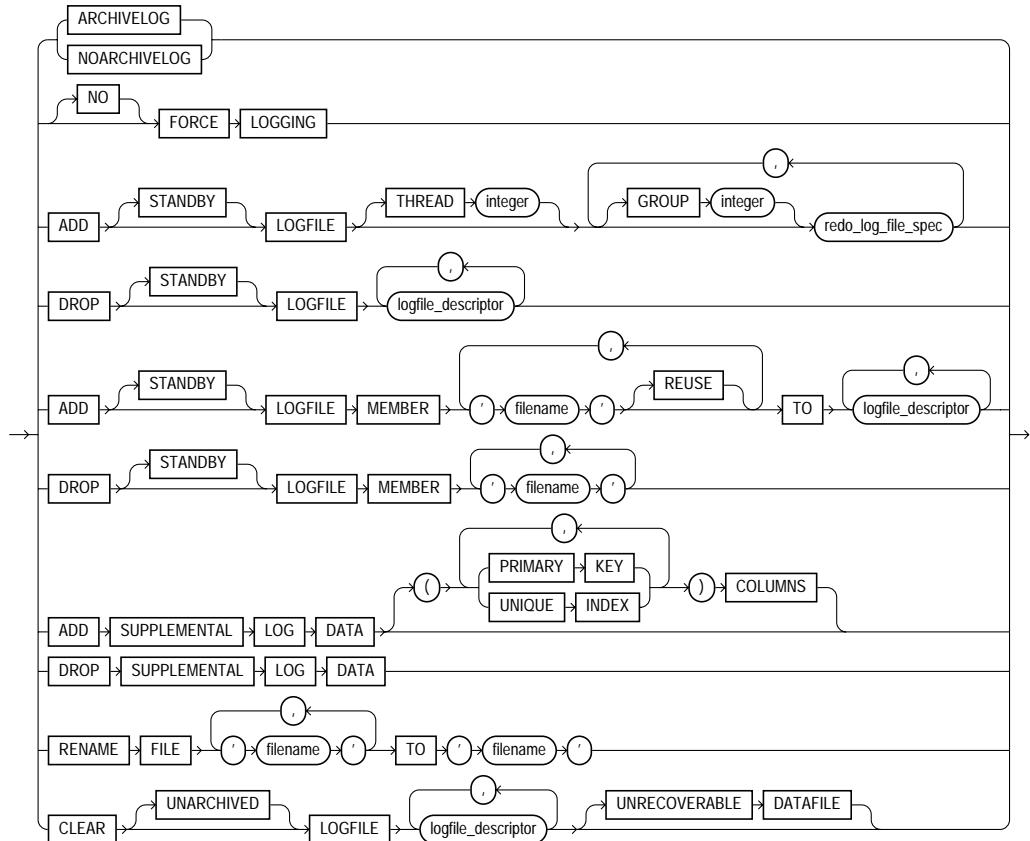


**general\_recovery****full\_database\_recovery****partial\_database\_recovery****parallel\_clause**

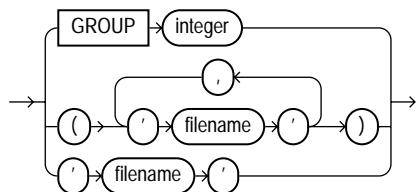
**managed\_standby\_recovery****recover\_clause****cancel\_clause****finish\_clause**

**database\_file\_clauses****autoextend\_clause****maxsize\_clause**

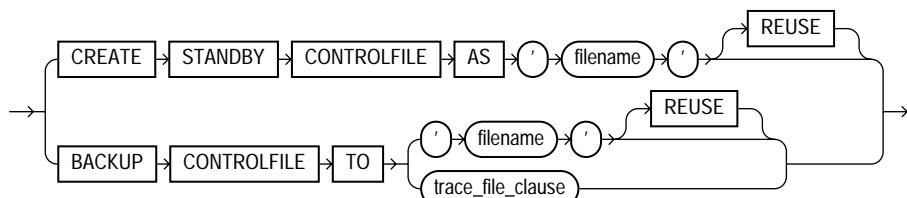
## logfile\_clauses



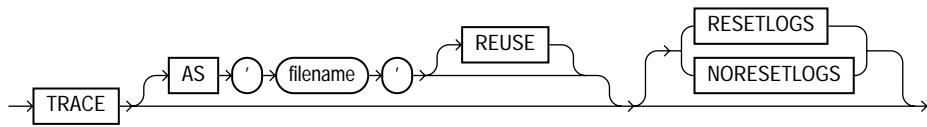
## logfile\_descriptor



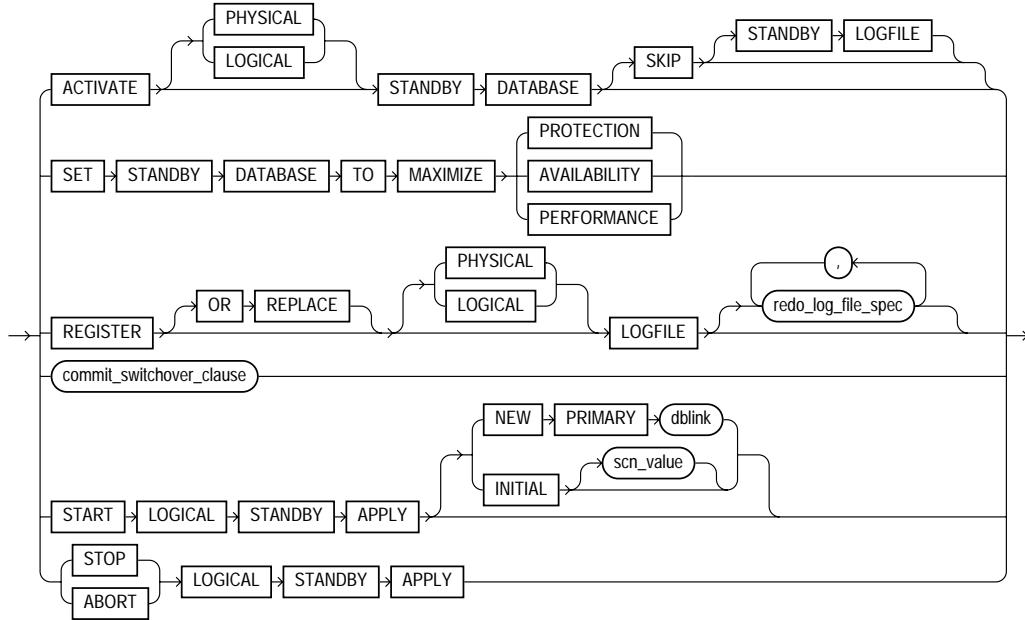
## **controlfile\_clauses**



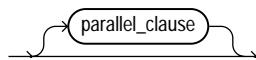
## trace\_file\_clause



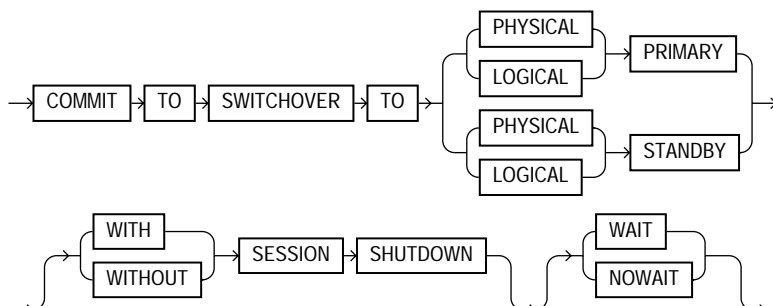
## standby\_database\_clauses

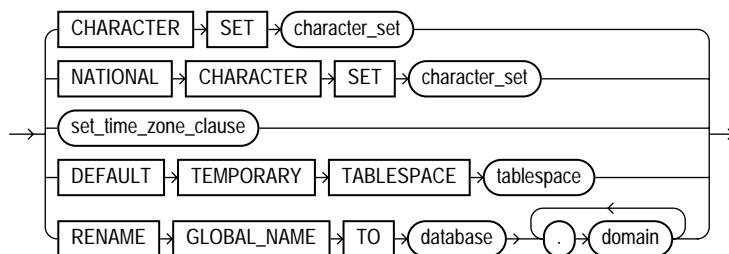
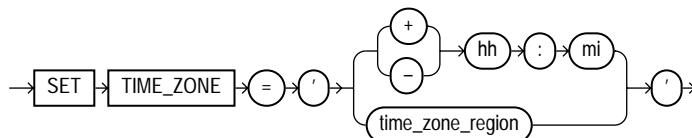
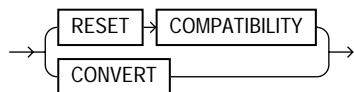
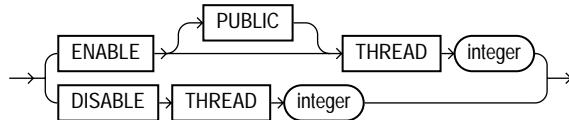


## standby\_database\_clauses2



## commit\_swtchover\_clause



**default\_settings\_clauses****set\_time\_zone\_clause****conversion\_clauses****redo\_thread\_clauses**

**DESCRIZIONE** Per poter modificare un database è necessario possedere il privilegio di sistema `ALTER DATABASE`.

*database* è il nome del database, di lunghezza pari o inferiore a otto caratteri.

Quando un database viene creato per la prima volta, viene montato (comando `MOUNT`) in modalità `EXCLUSIVE`, il che significa che nessuno può accedervi, eccetto colui che l'ha creato. Per consentire l'accesso al database da parte di più istanze (utenti, processi e così via) si utilizza `MOUNT PARALLEL`, sempre che sia stata installata l'opzione Real Application Clusters di Oracle.

Dopo aver montato un database, lo si apre (comando `OPEN`) e si azzerano i redo log (comando `RESETLOGS`), eliminando qualsiasi voce precedente. Quest'opzione viene utilizzata per ripristinare un database dopo un salvataggio parziale a causa di problemi tecnici verificatisi con i supporti fisici. L'opzione `NORESETLOGS` lascia la situazione invariata rispetto al momento in cui il database è stato aperto tramite `OPEN`.

La clausola `RECOVER` consente di ripristinare il database. Questo comando esegue un ripristino dai supporti fisici di un database che è andato perduto. Si possono eseguire più processi di ripristino per applicare le voci redo log ai file di dati per ciascuna istanza. Vedere `RECOVER`. Si può effettuare il `BACKUP` di un `CONTROLFILE` in un file specificato o in un file di trace.

I database STANDBY offrono un meccanismo per il failover automatizzato in caso di errore del database. Per ulteriori informazioni sulla creazione e la gestione di database standby si rimanda alla *Oracle9i Database Administrator's Guide*.

RENAME cambia il nome di un database o di un file di log esistente. Gli amministratori di database possono anche creare dei file di dati e dei file temporanei (rispettivamente con CREATE DATAFILE e CREATE TEMPFILE) e gestire gli incrementi tramite i quali i file di dati si estendono automaticamente. La clausola AUTOEXTEND può essere utilizzata per estendere, se necessario, in modo dinamico i file di dati, con incrementi di dimensione pari a NEXT, fino ad un massimo di MAXSIZE (o UNLIMITED). La clausola RESIZE può essere utilizzata per incrementare o decrementare la dimensione di un file di dati esistente. La clausola DATAFILE consente di mettere un file in linea (opzione ONLINE) o disconnetterlo (opzione OFFLINE). L'opzione CREATE consente di sostituire un file di dati vecchio con uno nuovo; in questo modo si può ricreare un file di dati perso, di cui non si dispone alcuna copia di salvataggio.

ARCHIVELOG e NOARCHIVELOG definiscono il modo in cui vengono utilizzati i file redo log. NOARCHIVELOG è l'opzione predefinita e significa che i file di redo possono essere riutilizzati senza che i relativi contenuti vengano salvati altrove. In questo modo, si assicura un recupero dell'istanza, fatta eccezione per problemi di funzionamento verificatisi con i supporti, quale un problema di crash del disco fisso. ARCHIVELOG impone l'archiviazione dei file di redo (solitamente su un altro disco o nastro), in modo che sia possibile effettuare il ripristino dei dati a fronte di un malfunzionamento del sistema. Questa modalità gestisce anche il ripristino dell'istanza.

ADD LOGFILE aggiunge un file o più file redo log al database. *filespec* specifica i nomi dei LOGFILE e le dimensioni: SIZE rappresenta il numero di byte riservati per questo file. Specificando il suffisso K, il valore viene moltiplicato per 1024; specificando M viene moltiplicato per 1048576. REUSE (senza SIZE) significa distruggere il contenuto di qualsiasi file con quel nome e assegnare il nome all'attuale database. SIZE con REUSE crea il file nel caso non esista ancora, altrimenti ne controlla la dimensione. SIZE da solo crea il file se non esiste, ma in caso contrario restituisce un errore. Il file di log viene assegnato a un thread, sia in modo esplicito tramite una clausola di thread, sia con un thread assegnato all'istanza corrente di Oracle. GROUP rappresenta una collection di file di log. Si può aggiungere questo gruppo di file di log elencandoli ed è possibile specificare il gruppo con un intero.

ADD LOGFILE MEMBER aggiunge nuovi file a un gruppo di file di log esistente, sia specificando l'intero GROUP, sia elencando tutti i file di log presenti nel gruppo.

DROP LOGFILE elimina un gruppo di file redo log esistente. DROP LOGFILE MEMBER elimina uno o più membri di un gruppo di file di log.

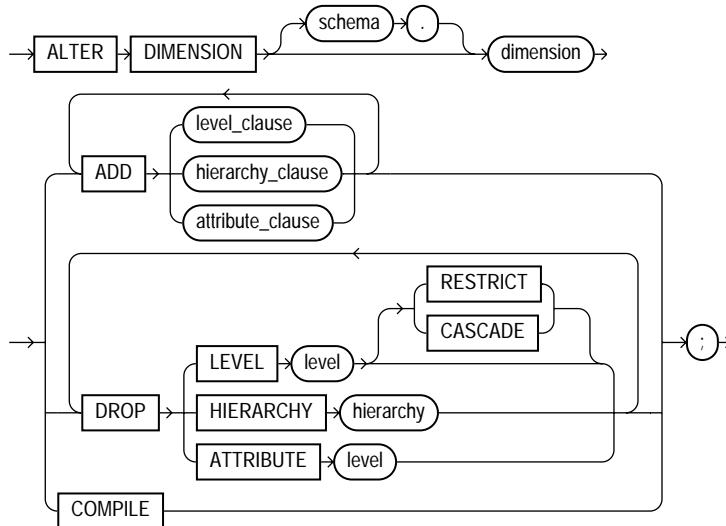
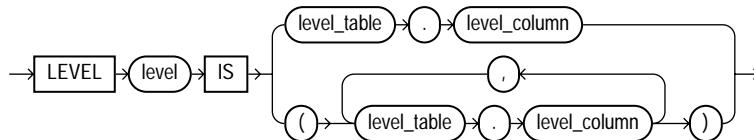
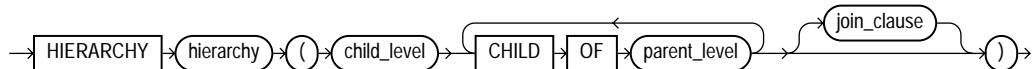
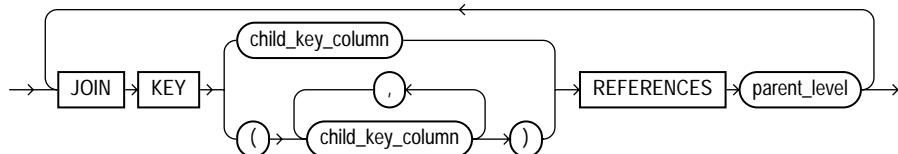
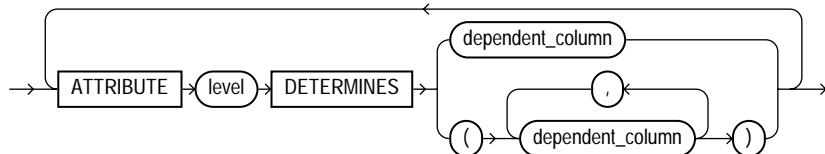
È possibile utilizzare RENAME GLOBAL\_NAME per modificare il nome del database. Se si specifica un dominio, occorre comunicare a Oracle la posizione del database all'interno della rete. È necessario modificare i riferimenti al database anche dai database remoti.

L'opzione RESET COMPATIBILITY, abbinata al parametro COMPATIBLE del file di inizializzazione dei parametri, viene utilizzata per tornare a una precedente versione di Oracle.

È possibile abilitare (ENABLE) o disabilitare (DISABLE) un THREAD. PUBLIC rende il thread disponibile per qualsiasi istanza che non richieda un thread specifico.

## ALTER DIMENSION

**VEDERE ANCHE** CREATE DIMENSION, DROP DIMENSION.

**SINTASSI****alter\_dimension****level\_clause****hierarchy\_clause****join\_clause****attribute\_clause**

**DESCRIZIONE** ALTER DIMENSION modifica gli attributi la gerarchia e i livelli di una dimensione esistente. Vedere CREATE DIMENSION.

### ESEMPIO

```
alter dimension TIME drop hierarchy MONTH_TO_DATE;
```

## ALTER FUNCTION

**VEDERE ANCHE** CREATE FUNCTION, DROP FUNCTION, Capitolo 29.

### SINTASSI

alter\_function



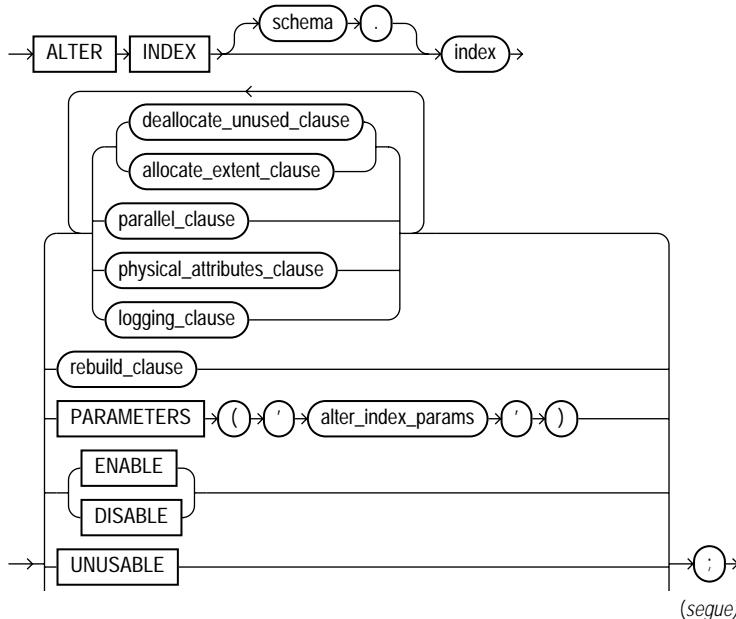
**DESCRIZIONE** ALTER FUNCTION ricompila una funzione PL/SQL. È possibile evitare sovraccarichi in fase di esecuzione e messaggi d'errore compilando esplicitamente in anticipo una funzione. Per modificare una funzione è necessario esserne il proprietario o possedere il privilegio di sistema ALTER ANY PROCEDURE.

## ALTER INDEX

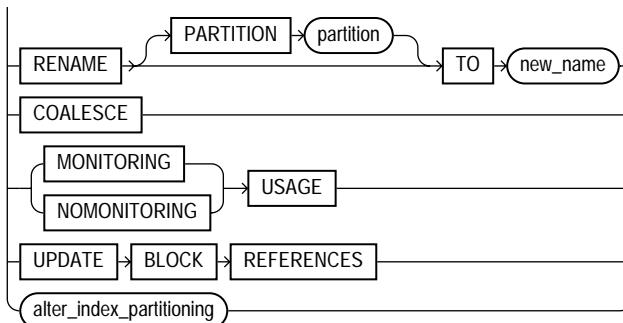
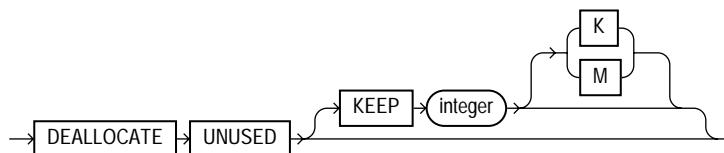
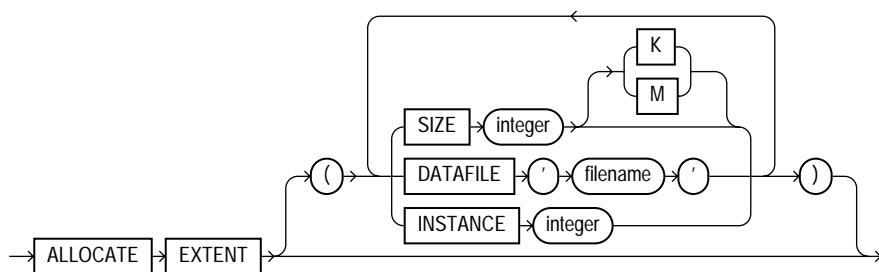
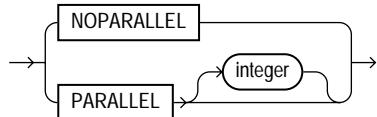
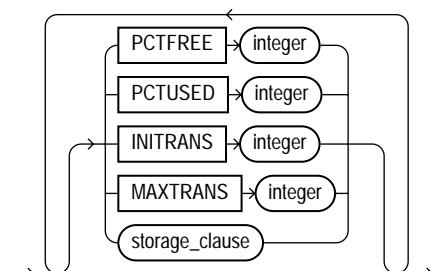
**VEDERE ANCHE** CREATE INDEX, DROP INDEX, STORAGE, Capitoli 18 e 20.

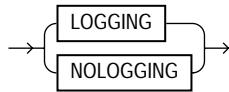
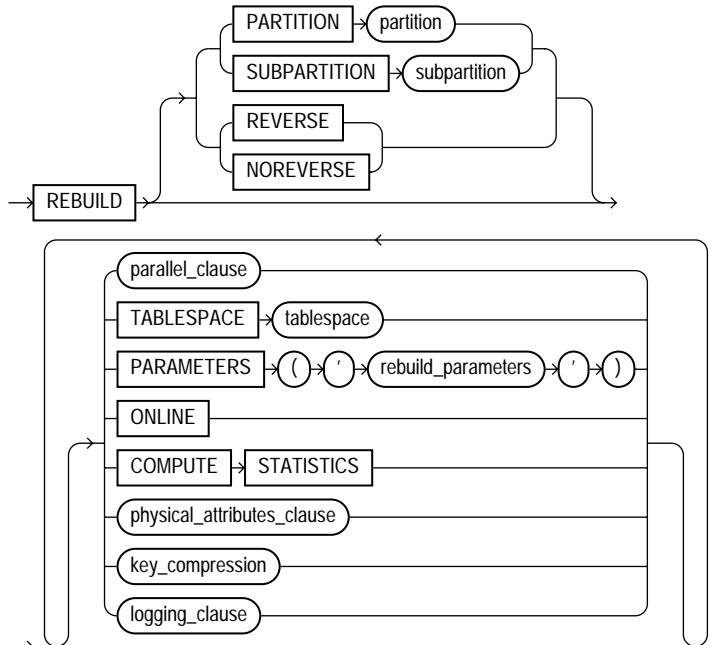
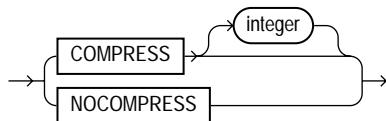
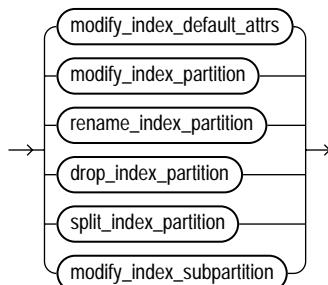
### SINTASSI

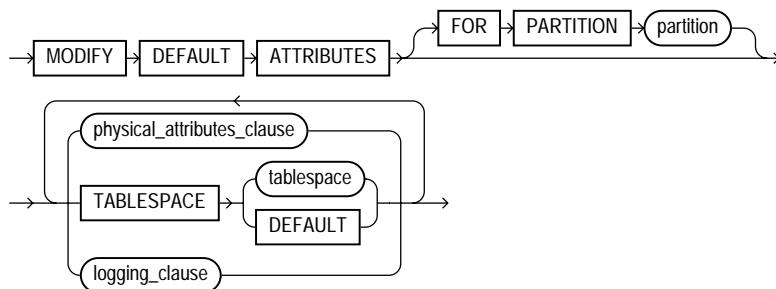
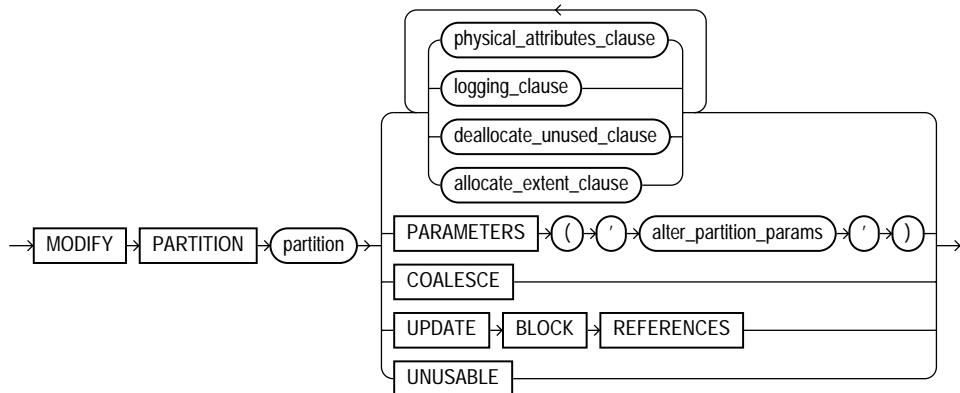
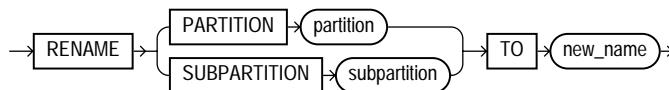
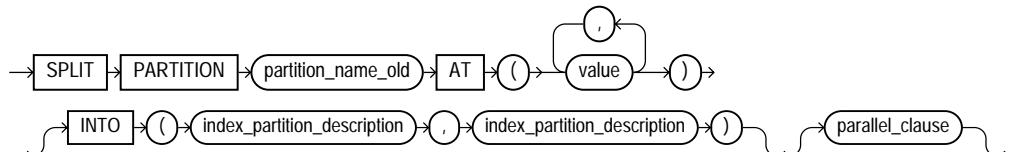
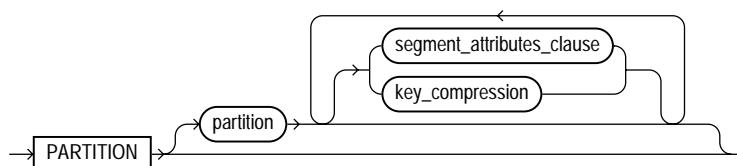
alter\_index



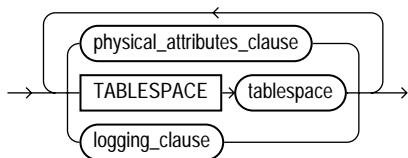
(continua)

`deallocate_unused_clause``allocate_extent_clause``parallel_clause``physical_attributes_clause`

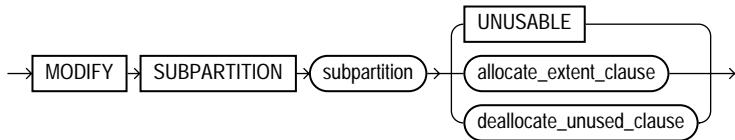
**logging\_clause****rebuild\_clause****key\_compression****alter\_index\_partitioning**

**modify\_index\_defaultAttrs****modify\_index\_partition****rename\_index\_partition****drop\_index\_partition****split\_index\_partition****index\_partition\_description**

### segment\_attributes\_clause



### modify\_index\_subpartition



**DESCRIZIONE** *utente* rappresenta l’utente in possesso dell’indice da modificare (gli altri utenti, per eseguire questo comando, devono possedere i privilegi DBA); *indice* è il nome di un indice esistente. Vedere CREATE TABLE per una descrizione di INITTRANS e MAXTRANS. Il valore predefinito per INITTRANS per quanto riguarda gli indici è 2; MAXTRANS è 255. STORAGE contiene clausole secondarie che sono descritte sotto STORAGE. Per poter creare un indice, è necessario disporre del privilegio INDEX sulla tabella e occorre inoltre essere il proprietario dell’indice oppure avere il privilegio di sistema CREATE ANY INDEX.

La clausola REBUILD del comando ALTER INDEX modifica le caratteristiche di registrazione di un indice esistente. REBUILD utilizza l’indice esistente come base per il nuovo indice. Vengono gestiti tutti i comandi di registrazione dell’indice, quali STORAGE (per l’allocazione di extent), TABLESPACE (per spostare l’indice in una nuova tablespace) e INITTRANS (per modificare il numero iniziale di voci).

È possibile modificare, rinominare, eliminare, suddividere o ricostruire partizioni dell’indice.

COMPRESS consente la compressione delle chiavi, eliminando l’occorrenza ripetuta di valori di colonna chiave per gli indici non unici. *intero* specifica la lunghezza del prefisso (numero di colonne prefisso da comprimere). Per default gli indici non sono compressi. Non è possibile comprimere un indice bitmap.

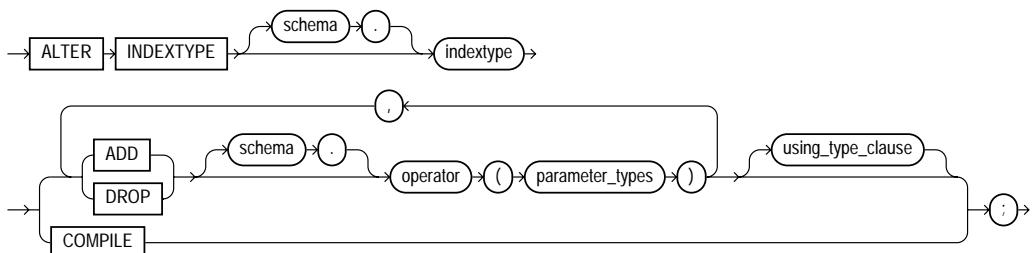
COMPUTE STATISTICS genera statistiche per l’ottimizzatore basato sui costi al momento della ricostruzione dell’indice.

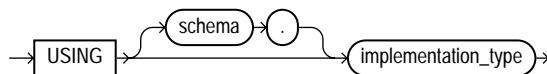
## ALTER INDEXTYPE

**VEDERE ANCHE** CREATE INDEXTYPE, DROP INDEXTYPE.

### SINTASSI

#### alter\_indextype



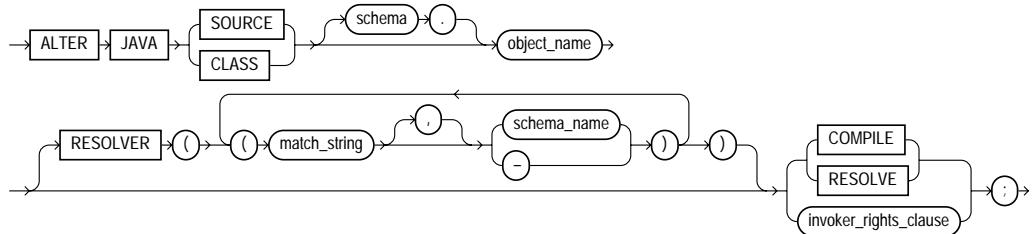
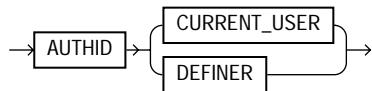
**using\_type\_clause**

**DESCRIZIONE** Utilizzare ALTER INDEXTYPE per aggiungere o rimuovere un operatore del tipo di indice, per modificare il tipo di implementazione, o per cambiare le proprietà del tipo di indice.

## ALTER JAVA

**VEDERE ANCHE** CREATE JAVA, DROP JAVA, Capitolo 34.

### SINTASSI

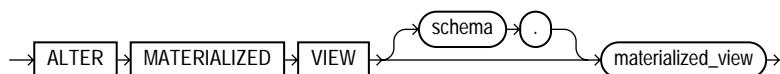
**alter\_java****invoker\_rights\_clause**

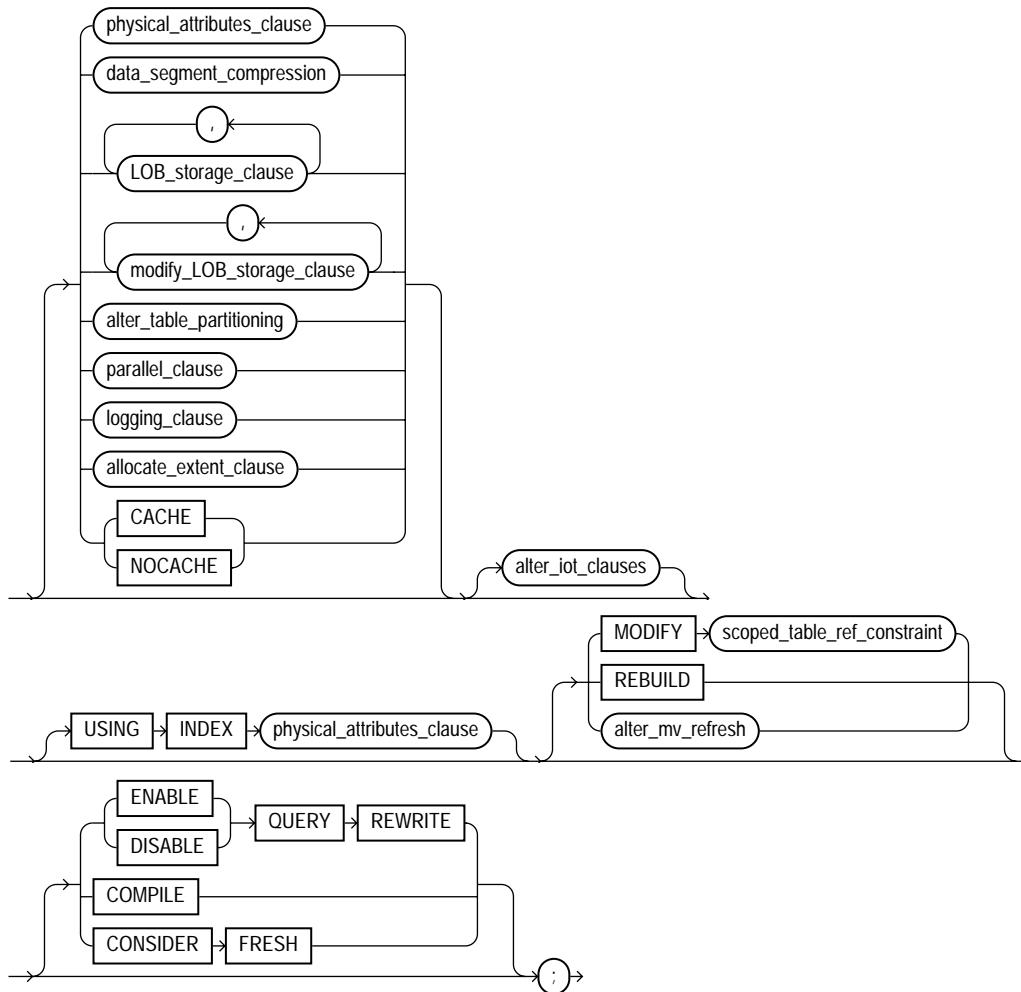
**DESCRIZIONE** ALTER JAVA ricompila un oggetto dello schema in sorgente Java. È anche possibile utilizzare questo comando per modificare l'autorizzazione imposta (proprietario o utente corrente). Per modificare gli oggetti dello schema dell'origine Java, è necessario essere proprietario dell'origine o della classe oppure essere in possesso del privilegio di sistema ALTER ANY PROCEDURE.

## ALTER MATERIALIZED VIEW

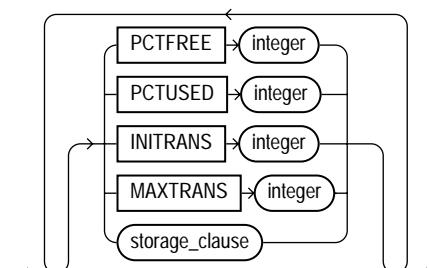
**VEDERE ANCHE** CREATE MATERIALIZED VIEW, DROP MATERIALIZED VIEW, MATERIALIZED VIEW, Capitolo 23.

### SINTASSI

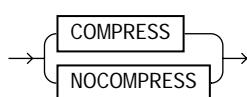
**alter\_materialized\_view**

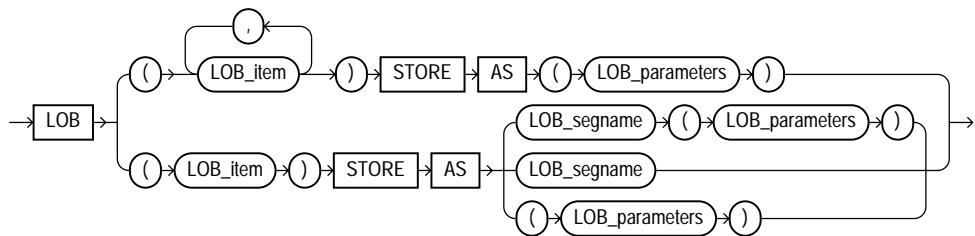
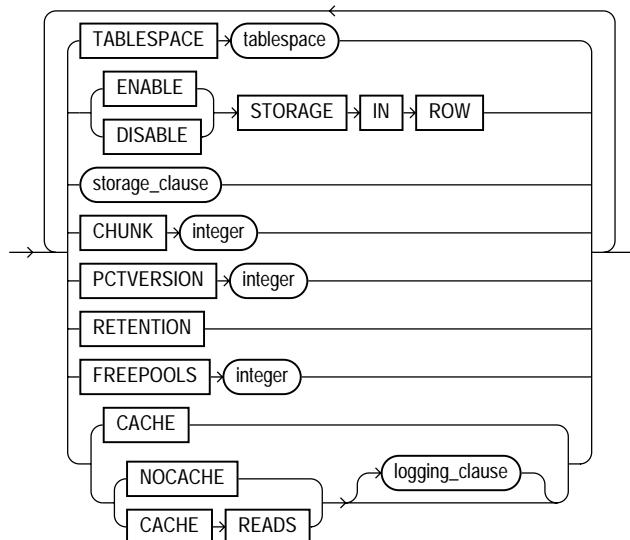
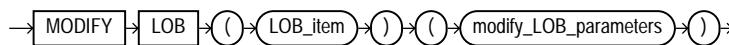
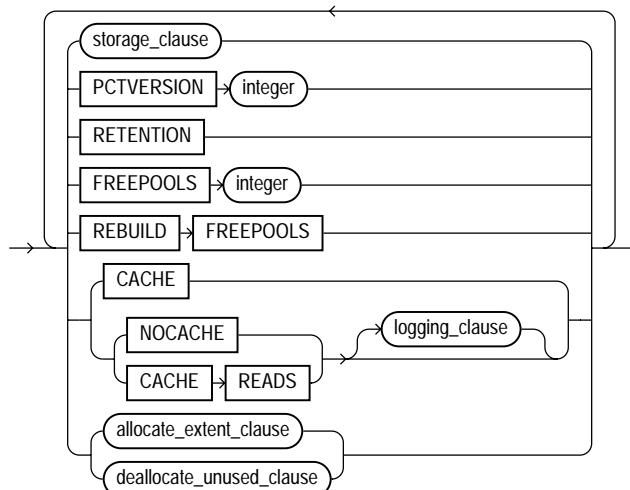


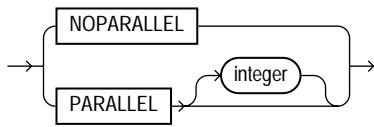
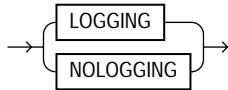
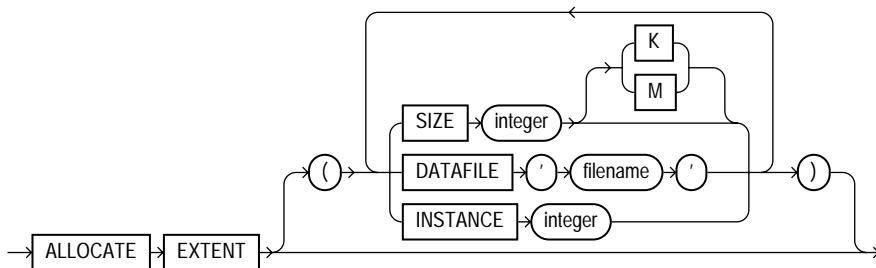
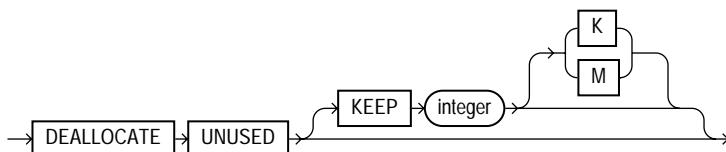
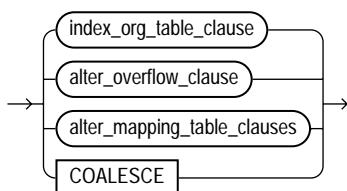
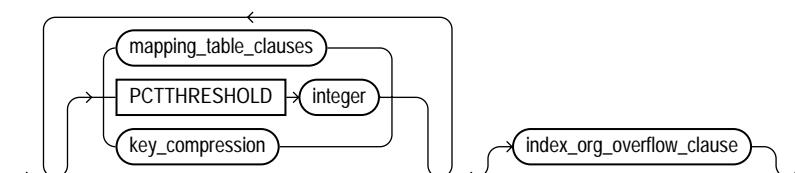
### **physical\_attributes\_clause**



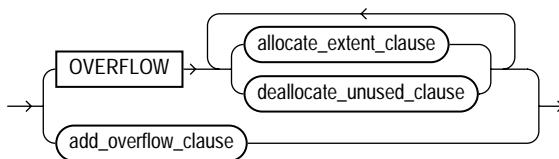
## **data segment compression**



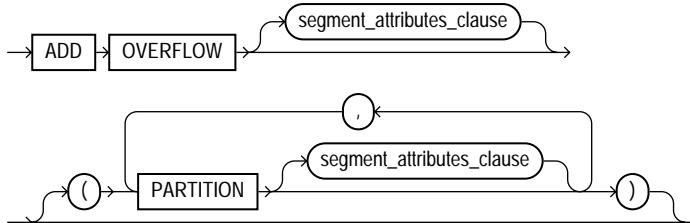
**LOB\_storage\_clause****LOB\_parameters****modify\_LOB\_storage\_clause****modify\_LOB\_parameters**

**parallel\_clause****logging\_clause****allocate\_extent\_clause****deallocate\_unused\_clause****alter\_iot\_clauses****index\_org\_table\_clause****index\_org\_overflow\_clause**

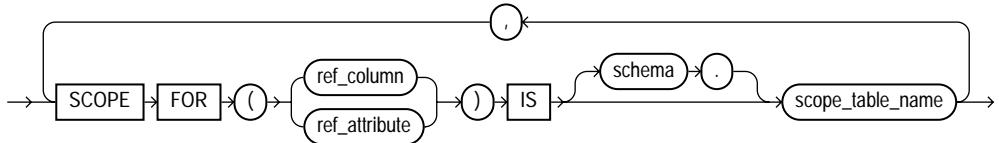
## **alter\_overflow\_clause**



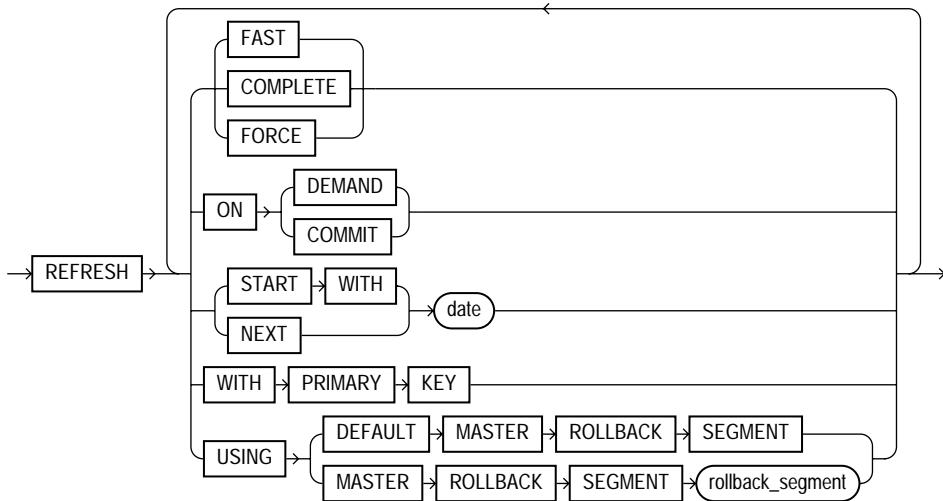
## **add\_overflow\_clause**



## scoped\_table\_ref\_constraint



## **alter\_mv\_refresh**



**DESCRIZIONE** La parola chiave `SNAPSHOT` è accettata in sostituzione di `MATERIALIZED VIEW` per garantire la compatibilità con le versioni precedenti.

**ALTER MATERIALIZED VIEW** consente di modificare le caratteristiche di memorizzazione o la modalità e il tempo con cui viene rinnovata una vista materializzata. È anche possibile attivare o disattivare la riscrittura delle query per le viste materializzate. Per modificare una vista materializzata è necessario esserne il proprietario o possedere il privilegio di sistema **ALTER**.

ANY SNAPSHOT o ALTER ANY MATERIALIZED VIEW. Vedere CREATE MATERIALIZED VIEW e il Capitolo 23 per ulteriori dettagli.

### ESEMPIO

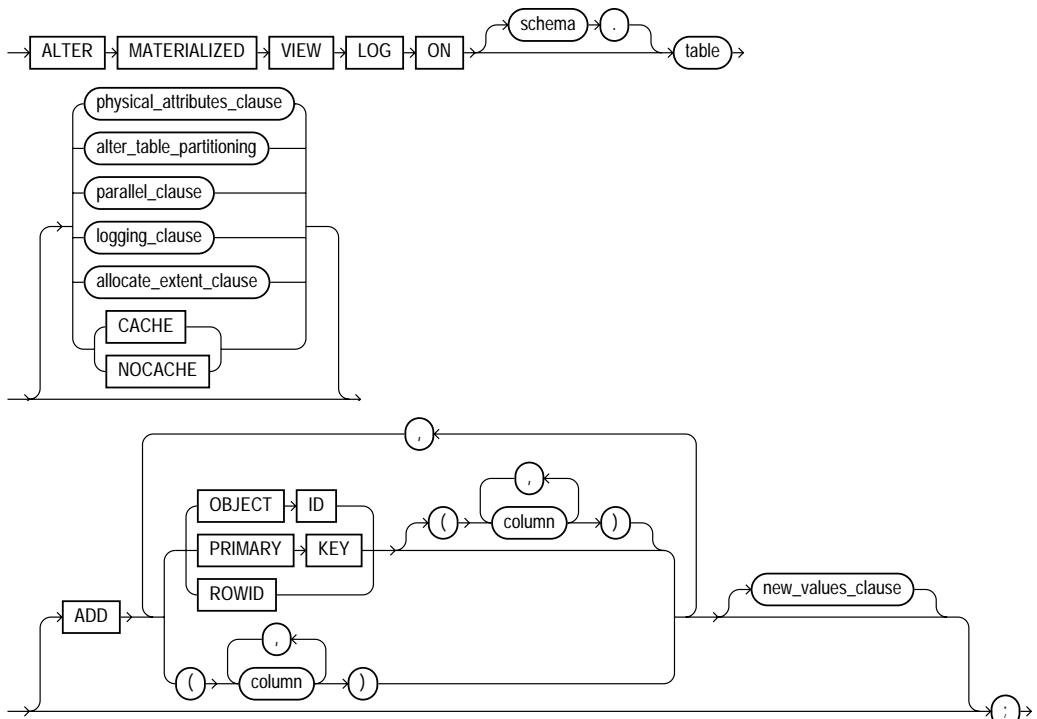
```
alter materialized view VENDITE_MENSILI
  enable query rewrite;
```

## ALTER MATERIALIZED VIEW LOG

**VEDERE ANCHE** CREATE MATERIALIZED VIEW, CREATE MATERIALIZED VIEW LOG, DROP MATERIALIZED VIEW LOG, Capitolo 23.

### SINTASSI

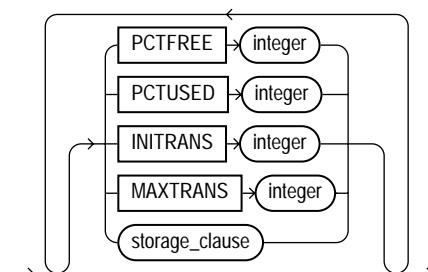
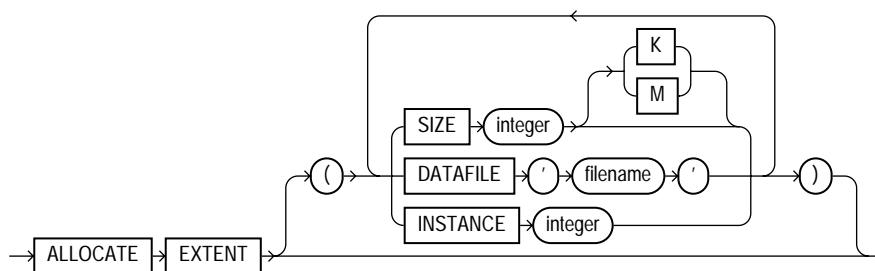
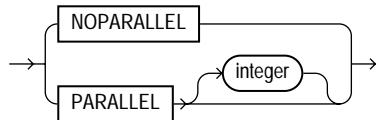
`alter_materialized_view_log`



`new_values_clause`

```

graph LR
    A[INCLUDING] --> B[NEW]
    A[EXCLUDING] --> B[NEW]
    B[NEW] --> C[VALUES]
  
```

**physical\_attributes\_clause****allocate\_extent\_clause****parallel\_clause**

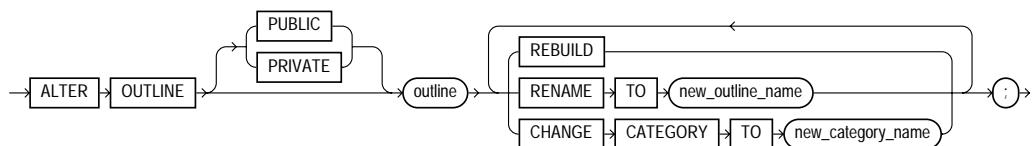
**DESCRIZIONE** La parola chiave `SNAPSHOT` è accettata in sostituzione di `MATERIALIZED VIEW` per garantire la compatibilità con le versioni precedenti.

`ALTER MATERIALIZED VIEW LOG` consente di modificare le caratteristiche di memorizzazione di un log delle viste materializzate. Per modificare il log, è necessario possedere il privilegio `ALTER` sulla tabella di log oppure disporre del privilegio di sistema `ALTER ANY TABLE`. Vedere `CREATE MATERIALIZED VIEW LOG` e il Capitolo 23.

## ALTER OUTLINE

**VEDERE ANCHE** `CREATE OUTLINE`, `DROP OUTLINE`.

### SINTASSI

**alter\_outline**

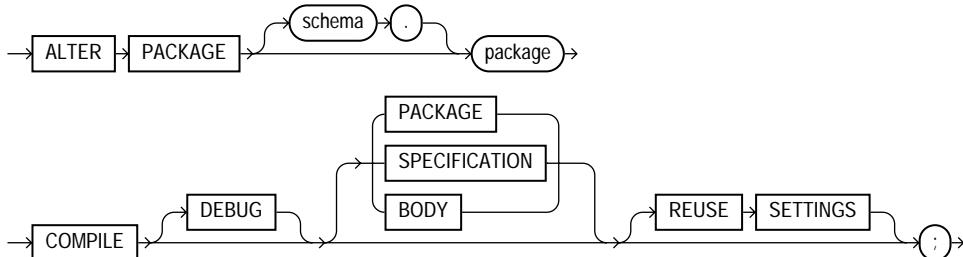
**DESCRIZIONE** È possibile utilizzare ALTER OUTLINE per rinominare una struttura memorizzata o per assegnarla a una categoria diversa. Le strutture memorizzate mantengono dei set di suggerimenti per le query eseguite in precedenza. Consultare l'*Oracle9i DBA Handbook*.

## ALTER PACKAGE

**VEDERE ANCHE** CREATE PACKAGE, DROP PACKAGE, Capitolo 29.

### SINTASSI

alter\_package



**DESCRIZIONE** ALTER PACKAGE ricompila le specifiche e/o il corpo di un package. Se le specifiche vengono ricompilate utilizzando PACKAGE, oltre alle specifiche viene ricompilato anche il corpo. Ricompilando le specifiche viene eseguita anche la ricompilazione delle procedure di riferimento; è possibile ricompilare il corpo (BODY) senza influenzare le specifiche. Per modificare un package è necessario esserne il proprietario oppure possedere il privilegio di sistema ALTER ANY PROCEDURE.

## ALTER PROCEDURE

**VEDERE ANCHE** CREATE PROCEDURE, DROP PROCEDURE, Capitolo 29.

### SINTASSI

alter\_procedure



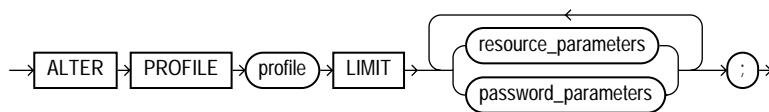
**DESCRIZIONE** ALTER PROCEDURE ricompila una procedura PL/SQL. Per evitare sovraccarichi in fase di esecuzione e messaggi d'errore è consigliabile compilare una procedura in anticipo, in modo esplicito. Per modificare una procedura, è necessario esserne il proprietario oppure possedere il privilegio di sistema ALTER ANY PROCEDURE.

## ALTER PROFILE

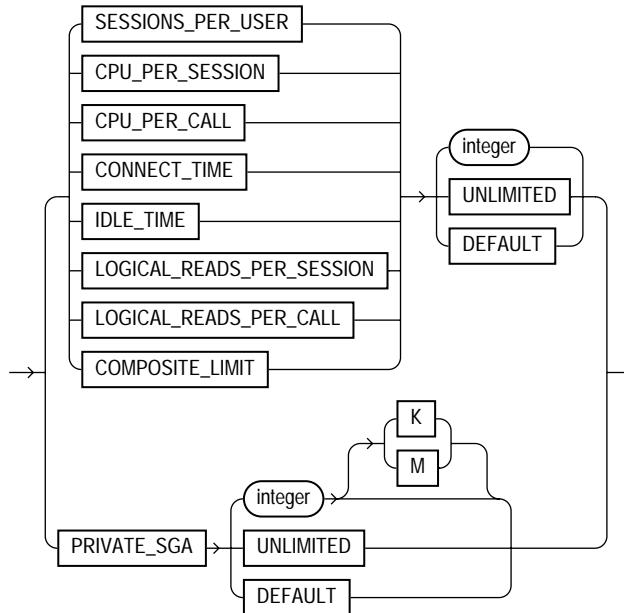
**VEDERE ANCHE** CREATE PROFILE, DROP PROFILE, Capitolo 19.

## SINTASSI

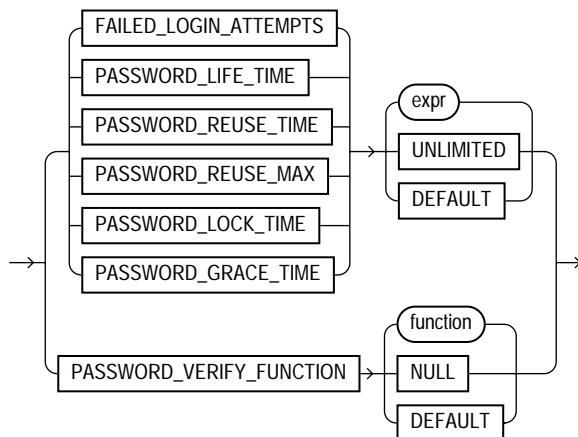
`alter_profile`



`resource_parameters`



`password_parameters`



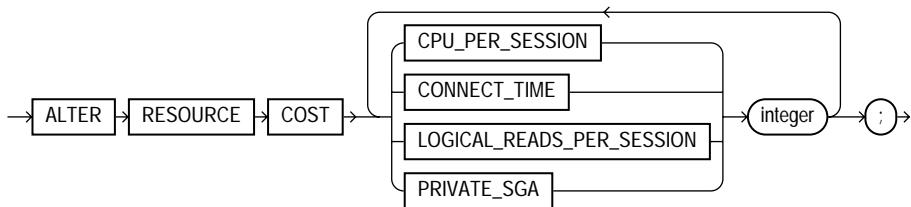
**DESCRIZIONE** `ALTER PROFILE` consente di modificare l'impostazione di un profilo. Vedere `CREATE PROFILE` per ulteriori dettagli.

## ALTER RESOURCE COST

**VEDERE ANCHE** CREATE PROFILE.

### SINTASSI

`alter_resource_cost`



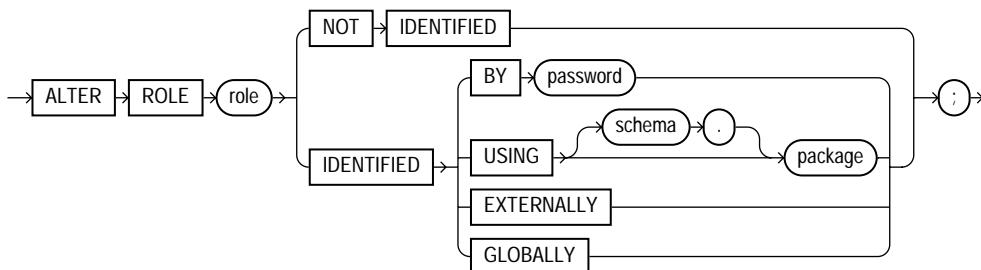
**DESCRIZIONE** La formula per il costo della risorsa calcola il costo totale delle risorse utilizzate in una sessione di Oracle. ALTER RESOURCE COST consente di assegnare un peso a diverse risorse. CPU\_PER\_SESSION rappresenta la quantità di CPU utilizzata in una sessione in centesimi di secondo. CONNECT\_TIME è il tempo trascorso all'interno di una sessione, in minuti. LOGICAL\_READS\_PER\_SESSION rappresenta il numero di blocchi di dati letti sia dalla memoria sia dal disco durante una sessione. PRIVATE\_SGA è il numero di byte contenuti nella SGA (System Global Area) privata, importante solo se si sta utilizzando l'architettura a server condiviso. Assegnando dei pesi, è possibile modificare la formula utilizzata per calcolare il costo della risorsa totale.

## ALTER ROLE

**VEDERE ANCHE** CREATE ROLE, DROP ROLE, Capitolo 19.

### SINTASSI

`alter_role`



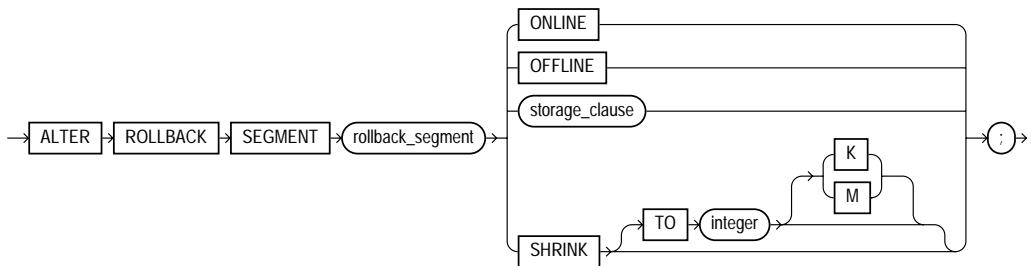
**DESCRIZIONE** ALTER ROLE consente di modificare un ruolo. Vedere CREATE ROLE e il Capitolo 19 per ulteriori dettagli.

## ALTER ROLLBACK SEGMENT

**VEDERE ANCHE** CREATE DATABASE, CREATE ROLLBACK SEGMENT, CREATE TABLESPACE, DROP ROLLBACK SEGMENT, STORAGE, Capitolo 40.

## SINTASSI

`alter_rollback_segment`



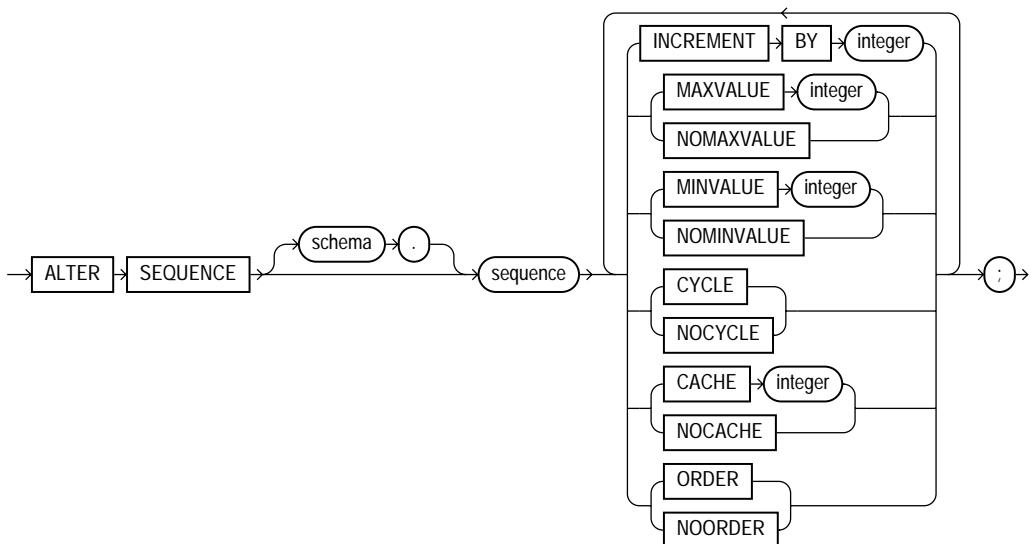
**DESCRIZIONE** *segmento* è il nome assegnato al segmento di rollback. STORAGE contiene delle clausole secondarie descritte alla voce STORAGE. Le opzioni INITIAL e MINEXTENTS non sono applicabili. Un segmento di rollback può essere reso ONLINE oppure OFFLINE nel corso dell'apertura del database. È possibile effettuare lo SHRINK del segmento a una dimensione specifica (per default, questa dimensione è quella specificata dal relativo OPTIMAL).

## ALTER SEQUENCE

**VEDERE ANCHE** AUDIT, CREATE SEQUENCE, DROP SEQUENCE, GRANT, NEXTVAL, REVOKE e CURRVAL alla voce PSEUDOCOLONNE, Capitolo 20.

## SINTASSI

`alter_sequence`



**DESCRIZIONE** Tutte queste opzioni influenzano l'utilizzo futuro di una sequenza esistente di nome *sequenza* (si osservi che start with non è disponibile; per modificare il valore con cui viene avviata una sequenza, è necessario eliminarla con DROP e ricrearla tramite CREATE).

Non può essere specificato un nuovo MAXVALUE inferiore all'attuale valore di CURRVAL per le sequenze in ordine crescente (ossia una sequenza che ha un numero positivo come valore INCREMENT BY). Analogamente, la stessa regola vale per MINVALUE per quanto riguarda le sequenze in ordine decrescente (caratterizzate da un valore INCREMENT BY negativo).

Per effettuare questa operazione è necessario possedere il privilegio ALTER per la sequenza o il privilegio di sistema DROP ANY SEQUENCE.

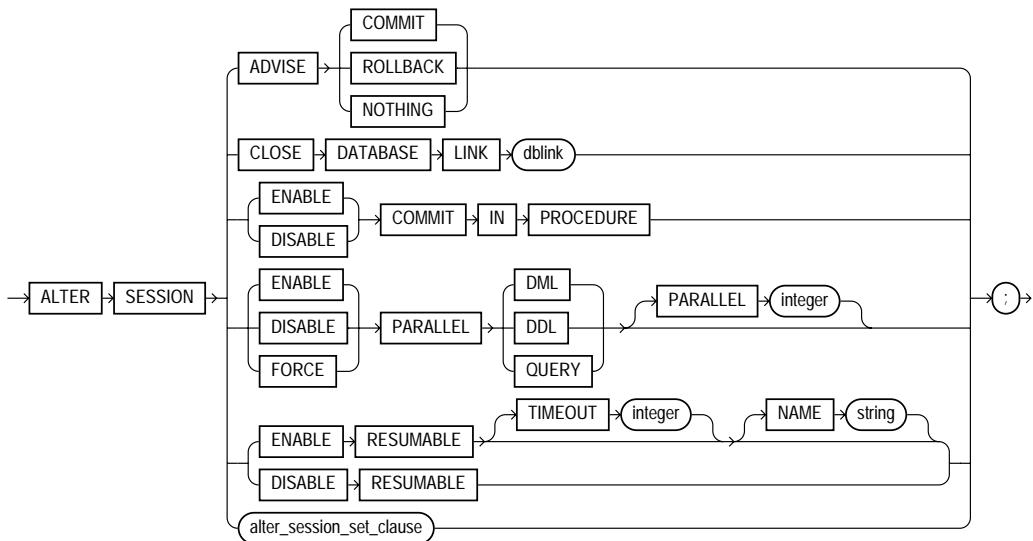
Tutte le altre funzionalità di ALTER SEQUENCE funzionano allo stesso modo che nel comando CREATE SEQUENCE, fatta eccezione per il fatto che vengono applicate a una sequenza esistente. Vedere CREATE SEQUENCE per ulteriori dettagli.

## ALTER SESSION

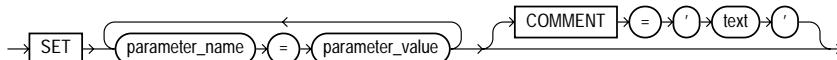
**VEDERE ANCHE** Oracle9i Performance Guide and Reference, Oracle9i Database Administrator's Guide.

### SINTASSI

alter\_session



alter\_session\_set\_clause



**DESCRIZIONE** Il comando ALTER SESSION modifica le impostazioni in atto per la sessione utente corrente. Impostando SQL\_TRACE al valore TRUE, vengono generate informazioni statistiche sulle prestazioni che riguardano l'elaborazione delle istruzioni SQL da parte di Oracle. Per disabilitare questa funzionalità è sufficiente impostare questo parametro al valore FALSE. Il parametro di inizializzazione SQL\_TRACE imposta il valore iniziale per questo report.

ADVISE consente di inviare una comunicazione di gestione della transazione ai database remoti coinvolti nelle transazioni distribuite. Per eliminare il collegamento di una sessione a un database remoto, è possibile chiudere (CLOSE) il collegamento al database (DATABASE LINK).

È possibile specificare se è possibile effettuare il COMMIT delle procedure, il grado in cui le operazioni DML vengono messe in parallelo, e se la ripresa delle operazioni è consentita. Se si abilitano le operazioni DML o DDL parallele per la propria sessione, occorre sempre specificare il suggerimento PARALLEL per le operazioni che devono essere eseguite in parallelo.

## ALTER SNAPSHOT

*Vedere ALTER MATERIALIZED VIEW.*

## ALTER SNAPSHOT LOG

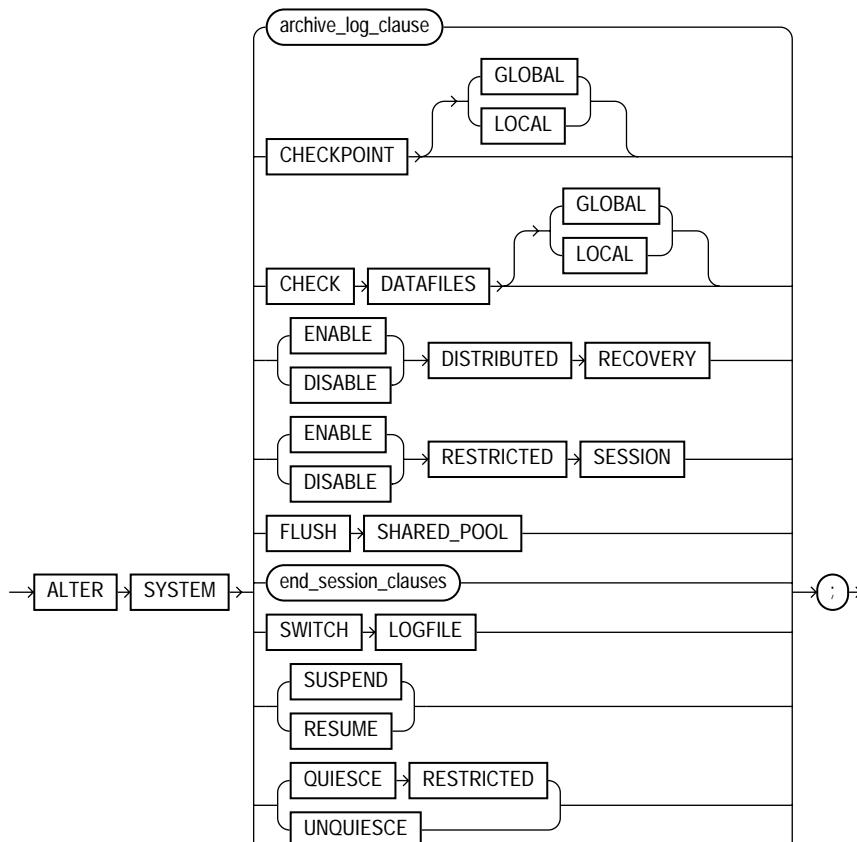
*Vedere ALTER MATERIALIZED VIEW LOG.*

## ALTER SYSTEM

**VEDERE ANCHE** ALTER SESSION, ARCHIVE\_LOG, CREATE PROFILE.

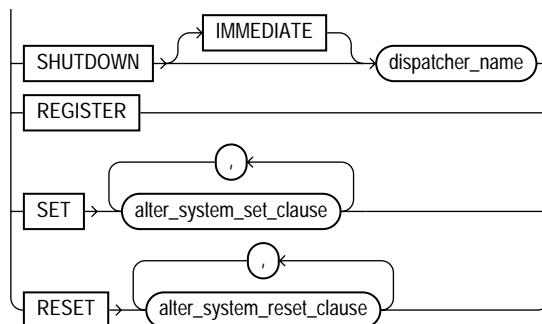
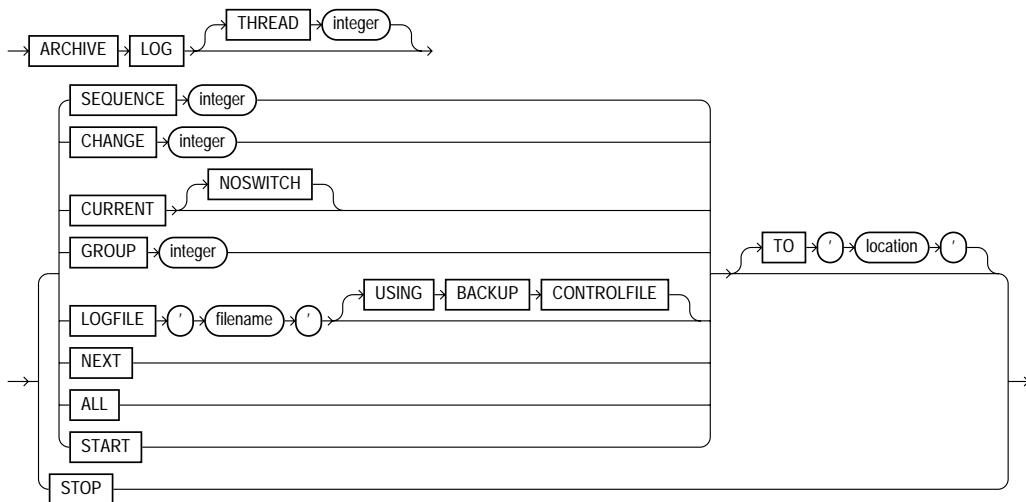
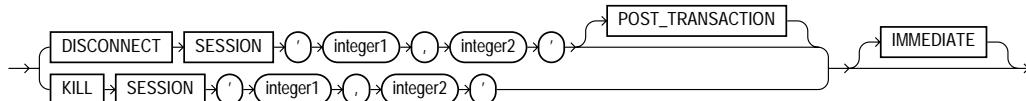
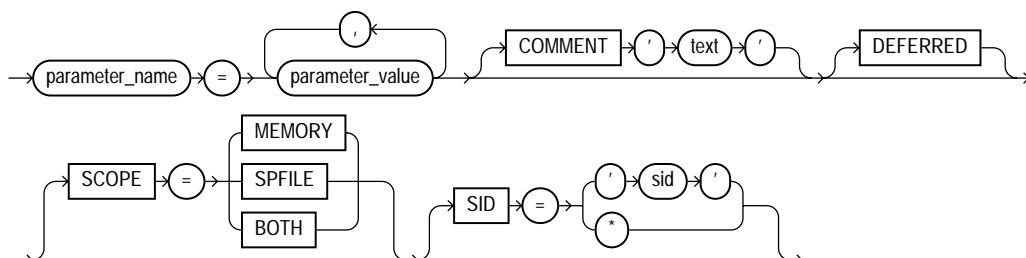
### SINTASSI

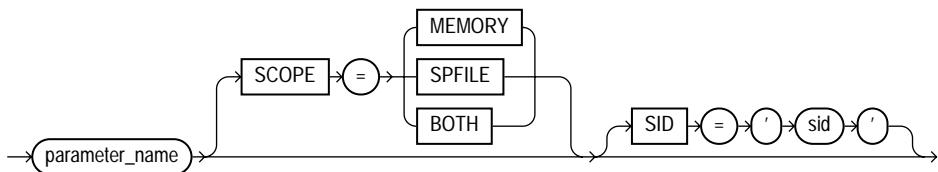
`alter_system`



(segue)

(continua)

**archive\_log\_clause****end\_session\_clauses****alter\_system\_set\_clause**

**alter\_system\_reset\_clause**

**DESCRIZIONE** ALTER SYSTEM consente di configurare la propria istanza di Oracle in molti modi. Le opzioni SET consentono di modificare molti valori dei parametri di inizializzazione mentre il database è in esecuzione. Se si utilizza un file di parametri di sistema, le modifiche saranno salvate e diverranno effettive al successivo avvio del database. Se invece si usa un file di parametri di inizializzazione (come init.ora), sarà necessario modificarlo in modo che i cambiamenti siano effettivi durante gli avvii successivi del database.

Con il comando ALTER SYSTEM è anche possibile cambiare file di log (SWITCH LOGFILE), costringendo Oracle a modificare i gruppi dei file di log.

CHECKPOINT effettua un checkpoint per tutte le istanze (GLOBAL) o per l'istanza di ORACLE dell'utente (LOCAL). CHECK DATAFILES verifica che ciascuna istanza (GLOBAL) o l'istanza di Oracle dell'utente (LOCAL) possa accedere a tutti i file di dati in linea.

ENABLE o DISABLE RESTRICTED SESSION abilita o disabilita una sessione limitata, accessibile solo agli utenti in possesso di quel privilegio di sistema.

ENABLE o DISABLE DISTRIBUTED RECOVERY abilita o disabilita, rispettivamente, questa opzione.

FLUSH SHARED\_POOL cancella tutti i dati dallo shared pool della SGA.

SUSPEND sospende l'attività del database; RESUME riattiva il database.

QUIESCE RESTRICTED colloca il database in uno stato coerente durante il quale solo SYS e SYSTEM, connessi come SYSDBA, possono eseguire operazioni. UNQUIESCE riporta il database alla sua modalità normale.

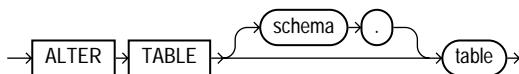
ARCHIVE LOG archivia manualmente i file redo log oppure abilita l'archiviazione automatica.

KILL SESSION termina una sessione sia con la colonna SID sia con la colonna SERIAL# della tabella V\$SESSION come identificativi della sessione. DISCONNECT SESSION, a differenza di KILL SESSION, offre l'opzione di forzare l'invio delle transazioni in sospeso della sessione prima della terminazione di quest'ultima.

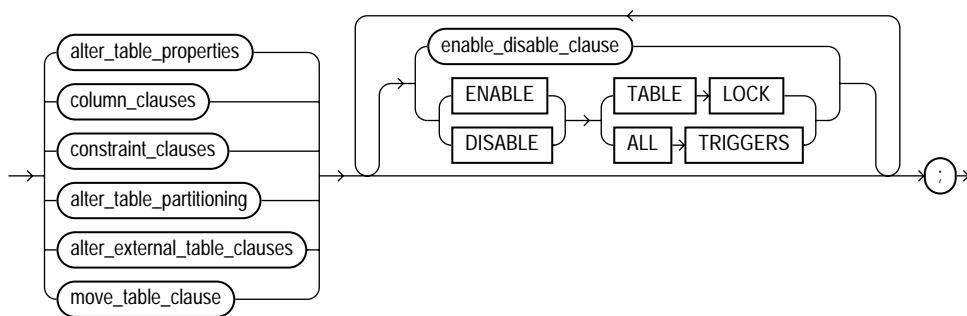
## ALTER TABLE

**VEDERE ANCHE** CREATE TABLE, DISABLE, DROP, ENABLE, Capitoli 18, 20, 31 e 32.

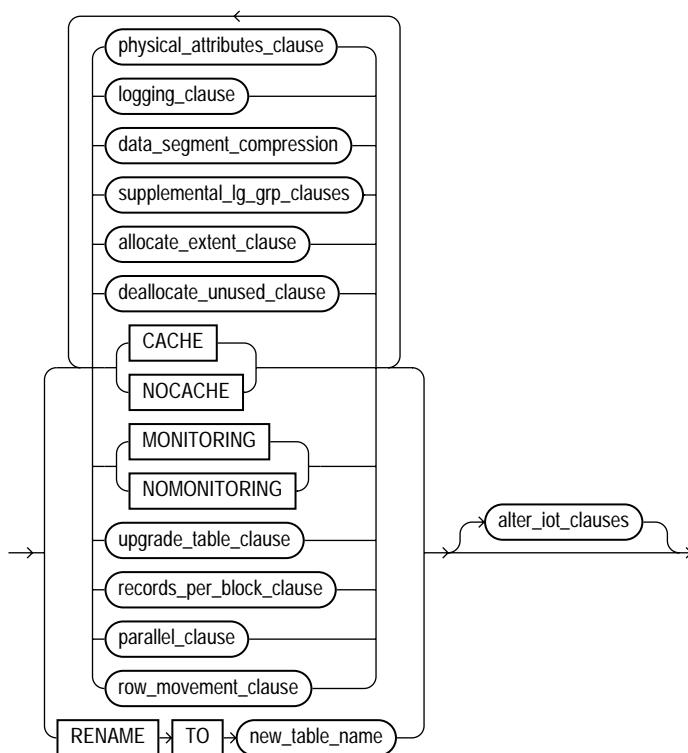
### SINTASSI

**alter\_table**

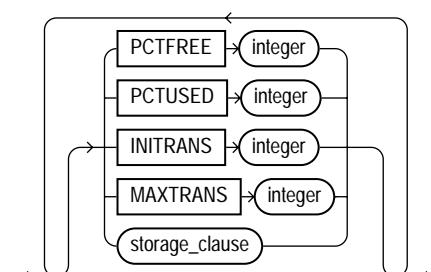
## alter\_table2



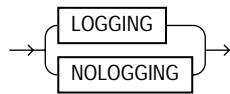
## alter\_table\_properties



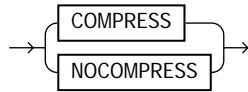
## physical\_attributes\_clause



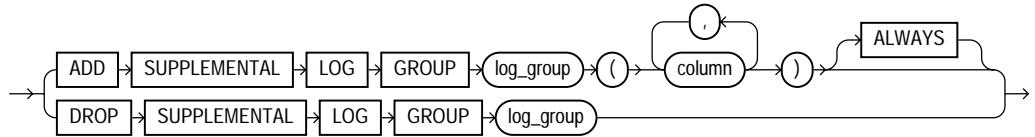
## logging\_clause



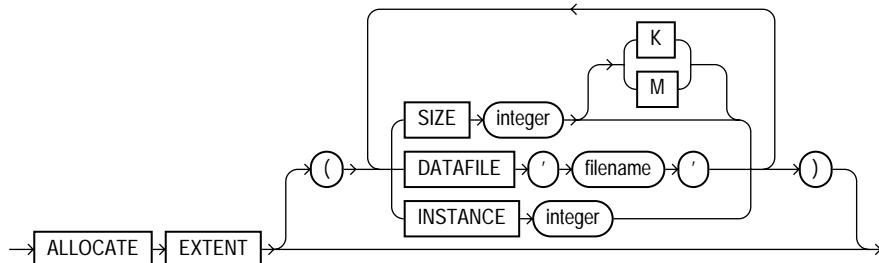
### **data\_segment\_compression**



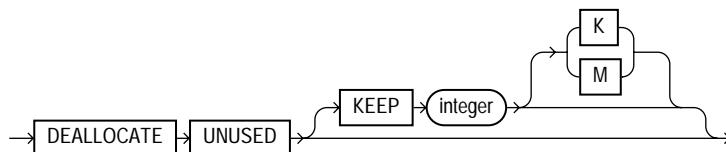
### **supplemental\_lg\_grp\_clauses**



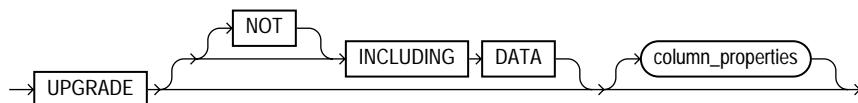
## **allocate\_extent\_clause**



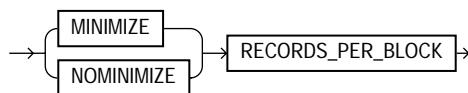
## **deallocate\_unused\_clause**

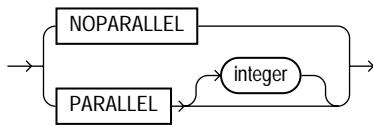
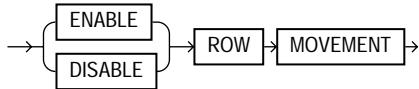
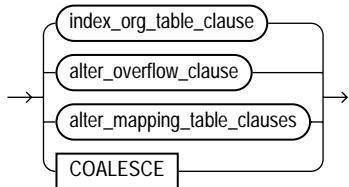
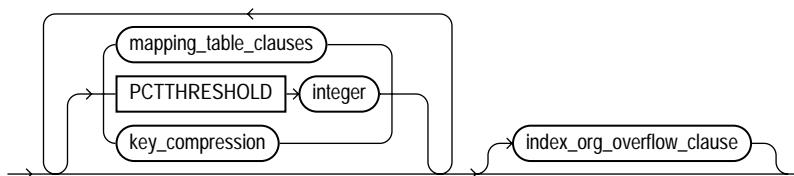
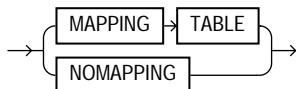
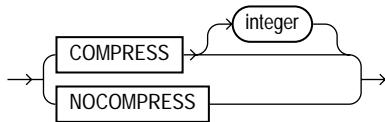
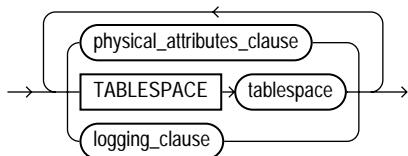


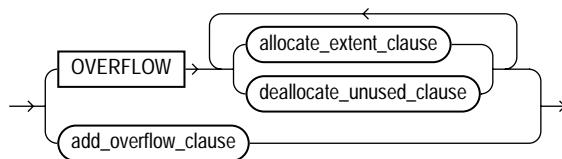
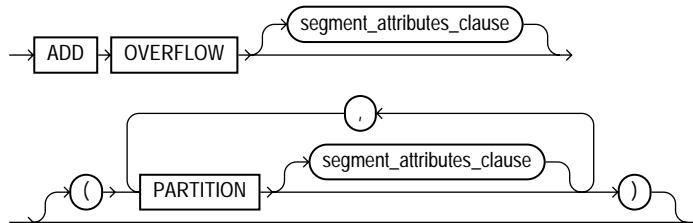
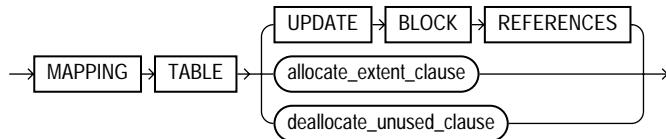
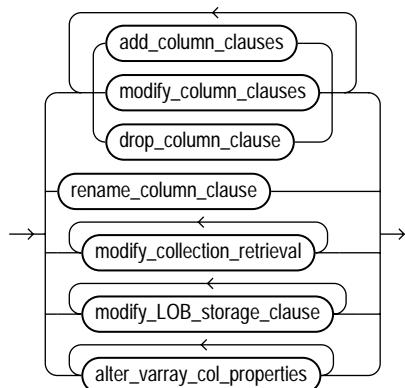
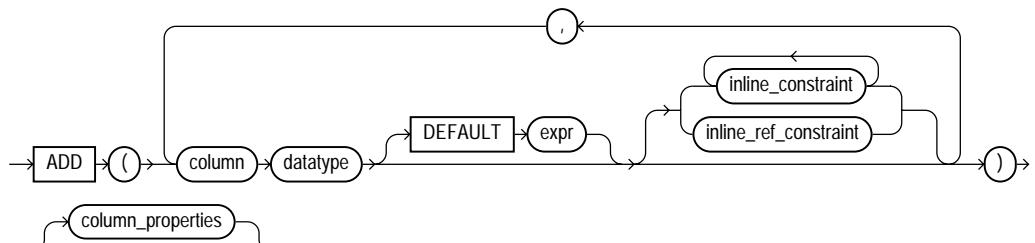
## upgrade\_table\_clause

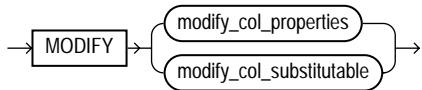
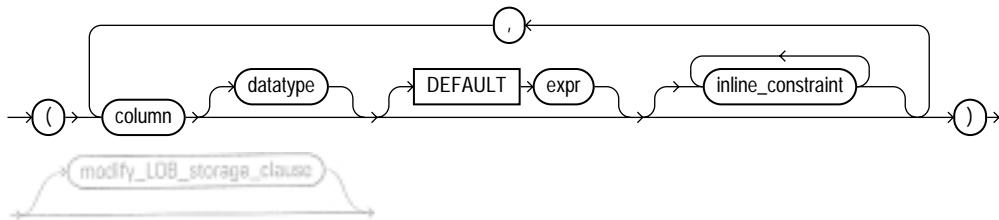
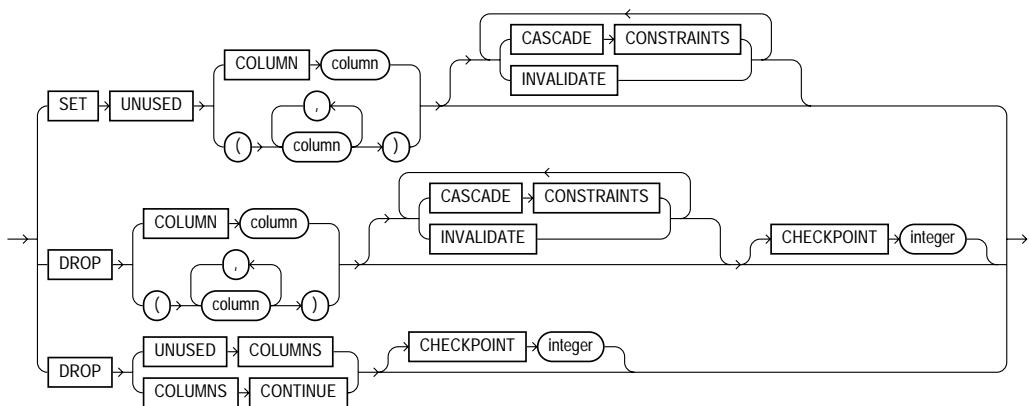


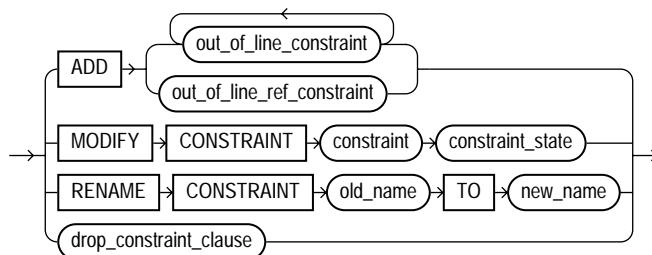
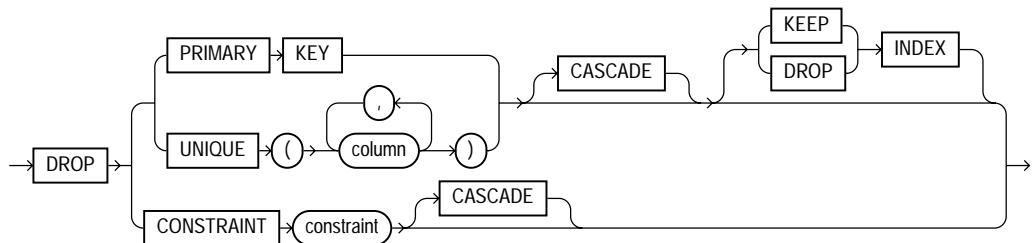
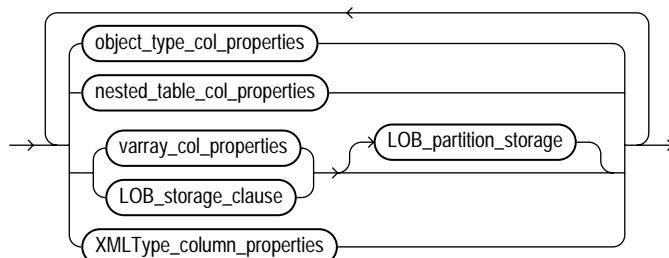
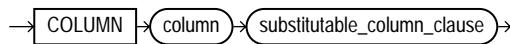
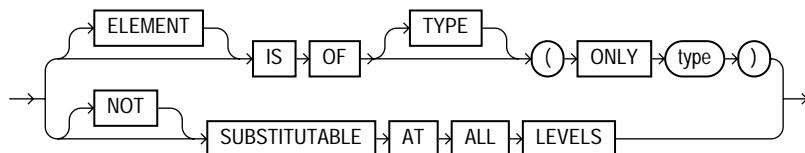
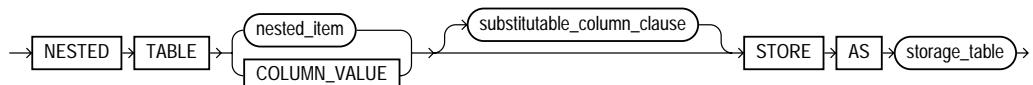
`records_per_block_clause`

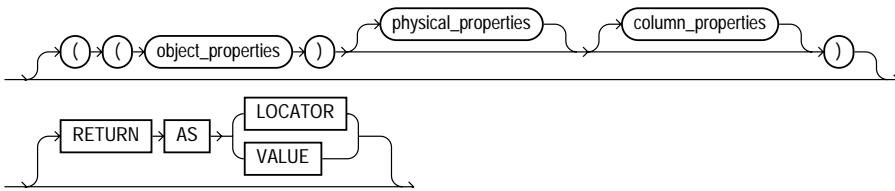
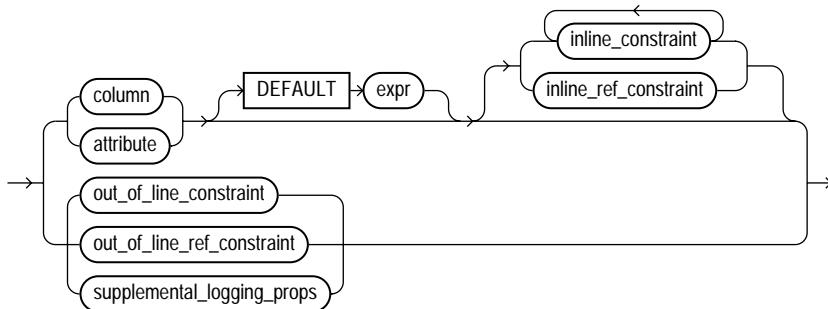
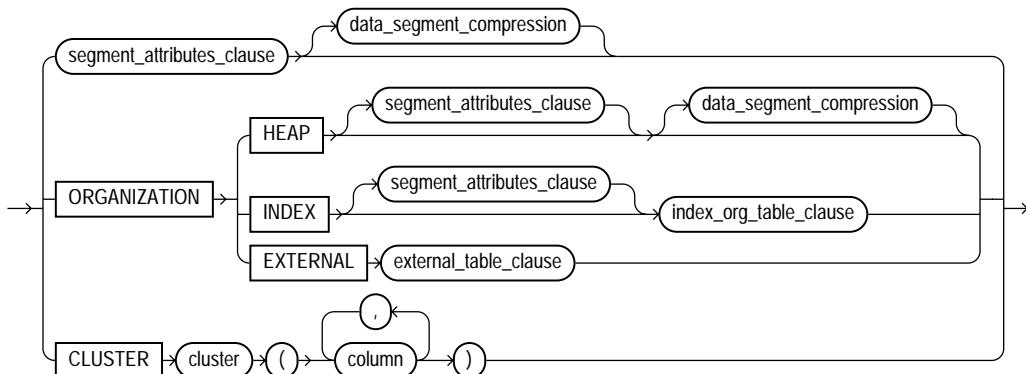
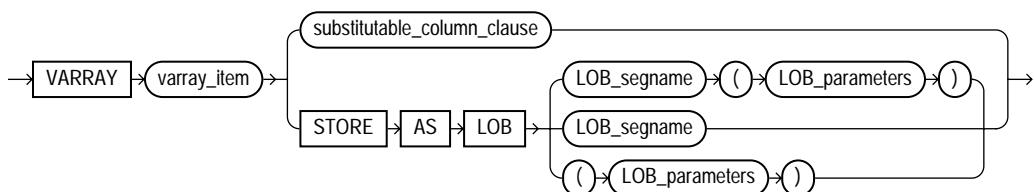


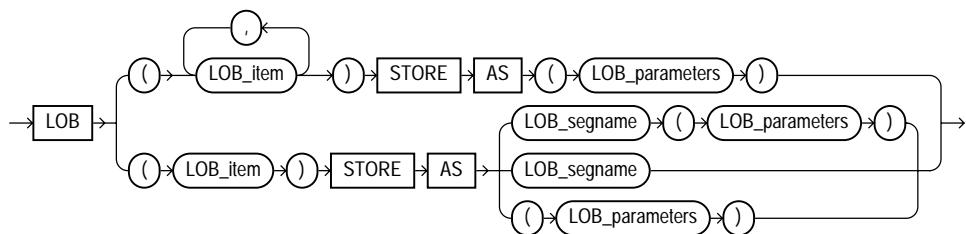
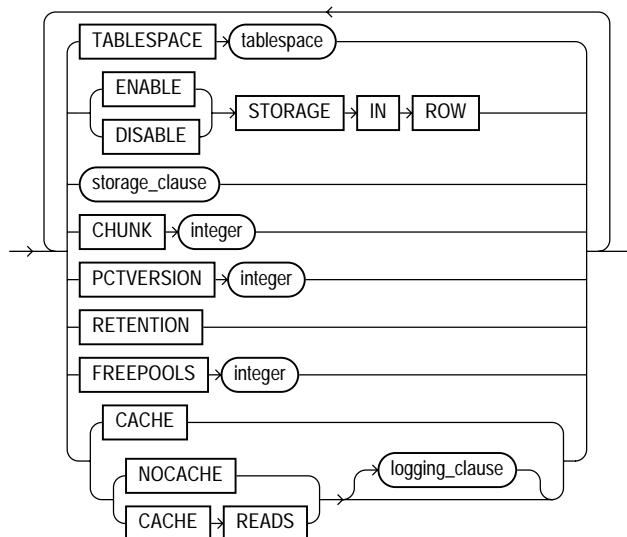
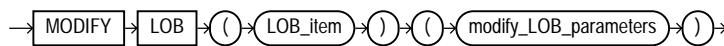
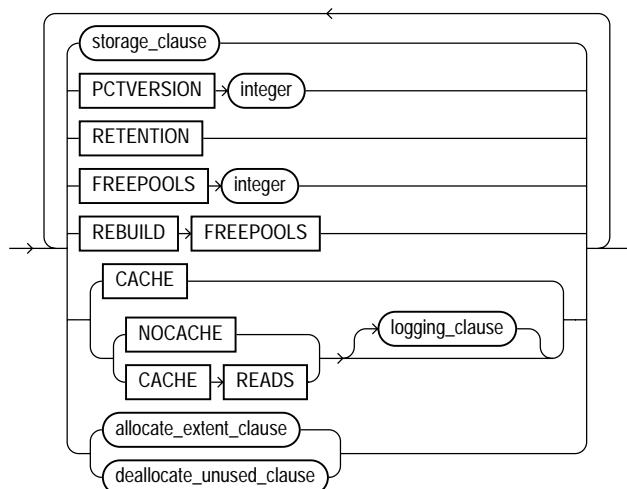
**parallel\_clause****row\_movement\_clause****alter\_iot\_clauses****index\_org\_table\_clause****mapping\_table\_clauses****key\_compression****index\_org\_overflow\_clause****segment\_attributes\_clause**

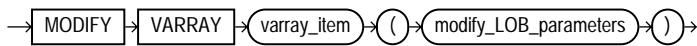
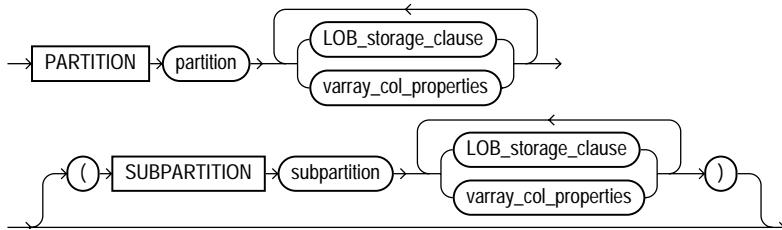
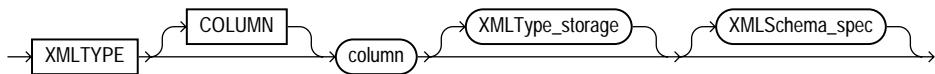
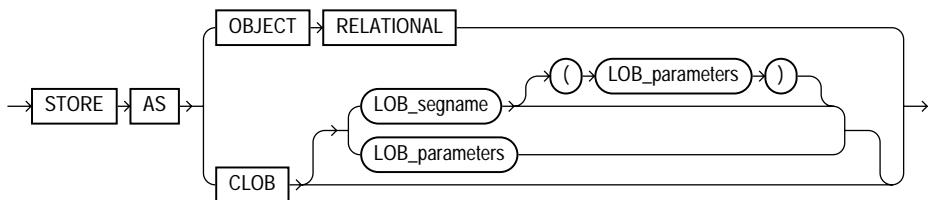
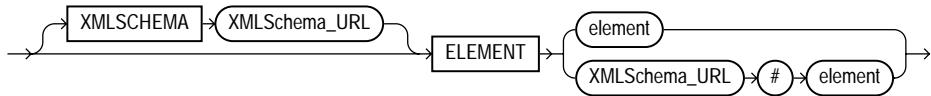
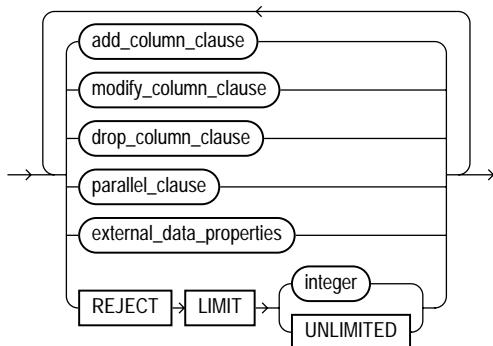
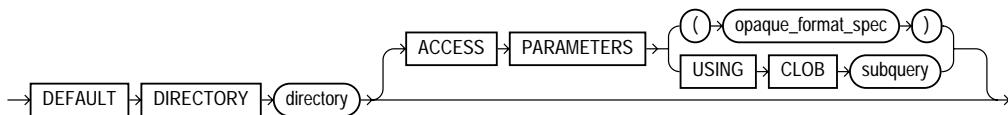
**alter\_overflow\_clause****add\_overflow\_clause****alter\_mapping\_table\_clause****column\_clauses****add\_column\_clause**

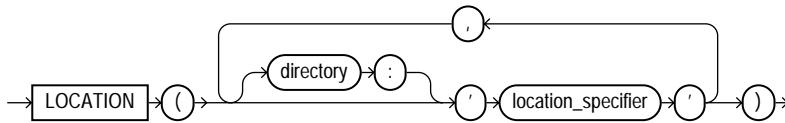
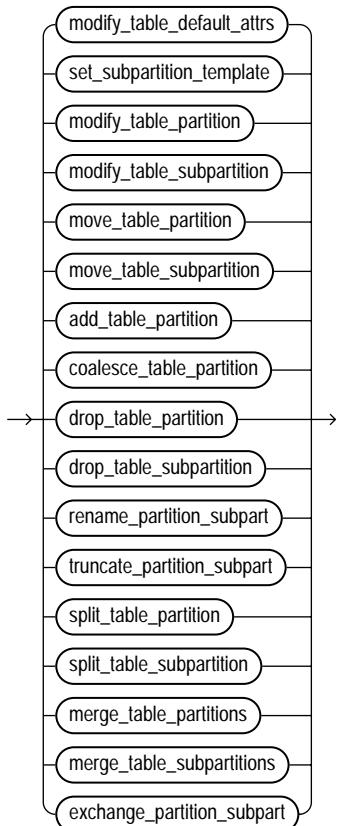
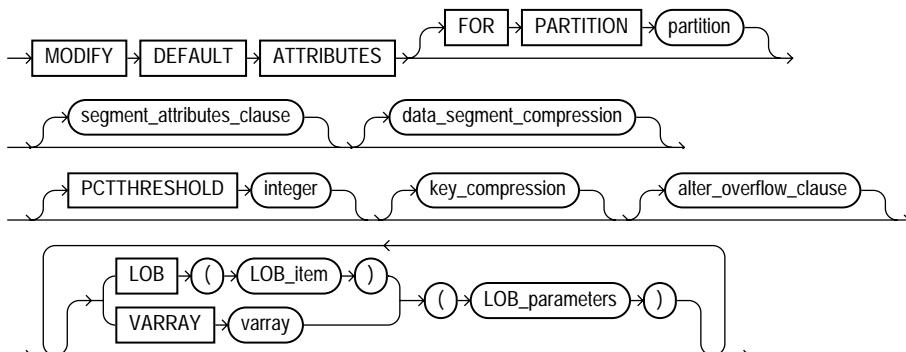
**modify\_column\_clause****modify\_col\_properties****modify\_col\_substitutable****drop\_column\_clause****rename\_column\_clause****modify\_collection\_retrieval**

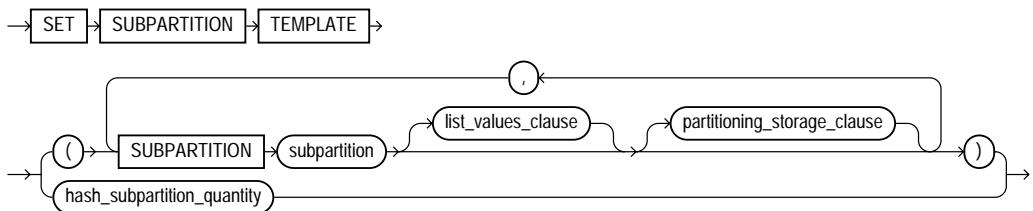
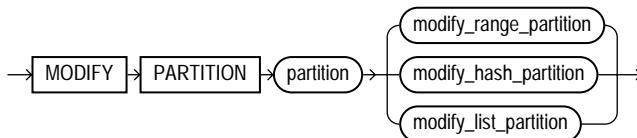
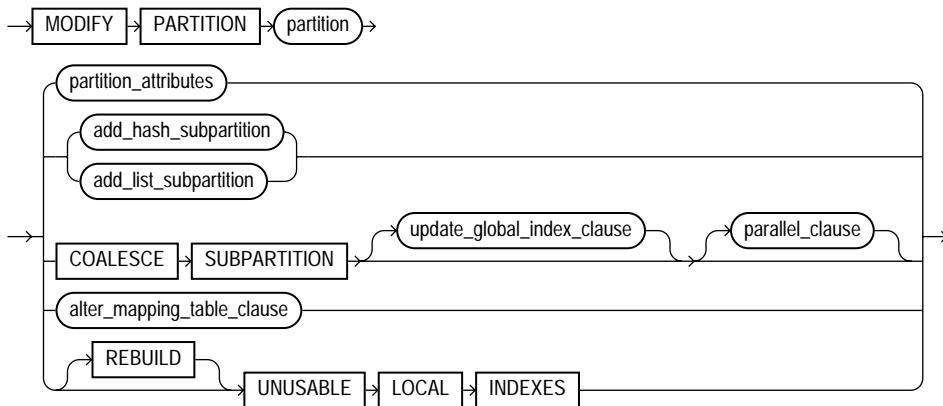
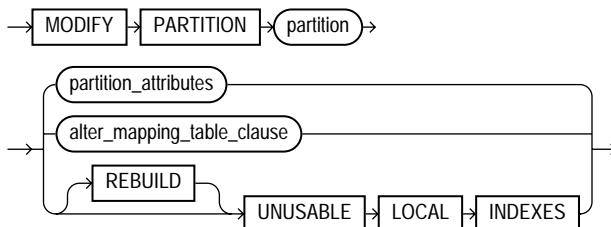
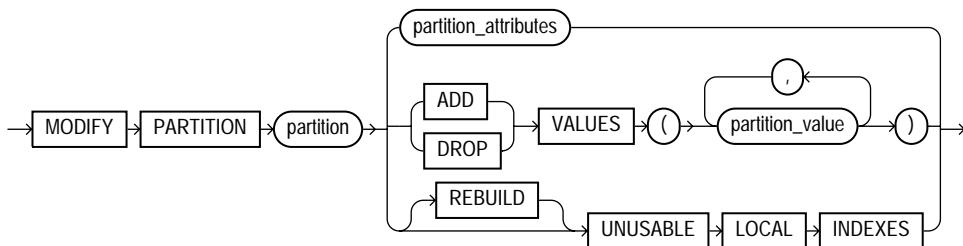
**constraint\_clauses****drop\_constraint\_clause****column\_properties****object\_type\_col\_properties****substitutable\_column\_clause****nested\_table\_col\_properties**

**object\_properties****supplemental\_logging\_props****physical\_properties****varray\_col\_properties**

**LOB\_storage\_clause****LOB\_parameters****modify\_LOB\_storage\_clause****modify\_LOB\_parameters**

**alter\_varray\_col\_properties****LOB\_partition\_storage****XMLType\_column\_properties****XMLType\_storage****XMLSchema\_spec****alter\_external\_table\_clauses****external\_data\_properties**

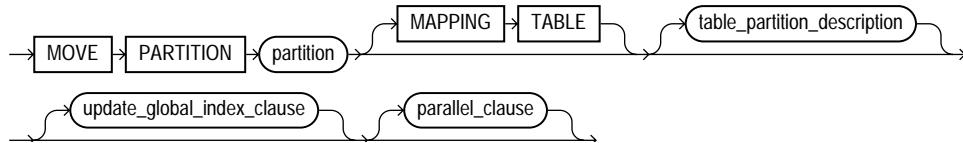
**alter\_table\_partitioning****modify\_table\_defaultAttrs**

**set\_subpartition\_template****modify\_table\_partition****modify\_range\_partition****modify\_hash\_partition****modify\_list\_partition**

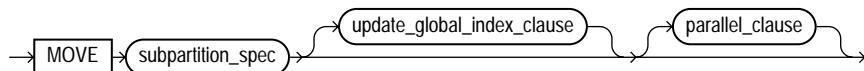
**modify\_table\_subpartition**



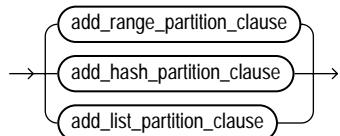
**move\_table\_partition**



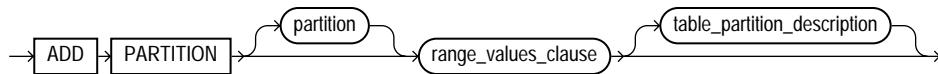
**move\_table\_subpartition**



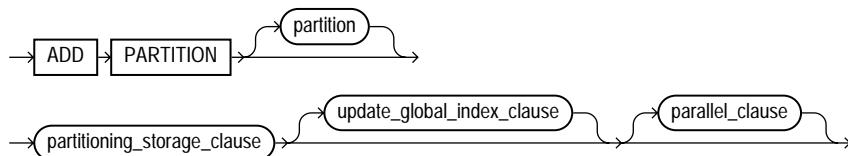
**add\_table\_partition**



**add\_range\_partition\_clause**



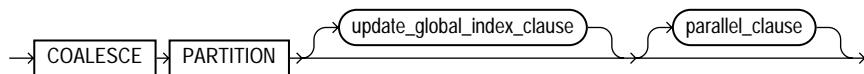
**add\_hash\_partition\_clause**



**add\_list\_partition\_clause**

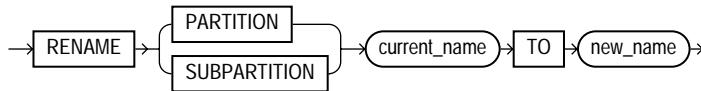
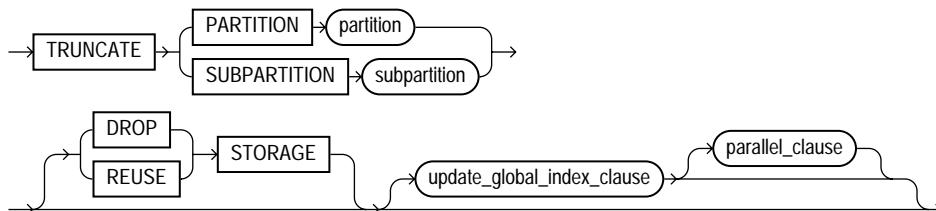
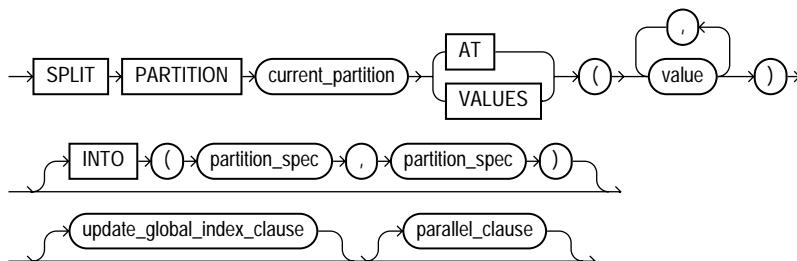
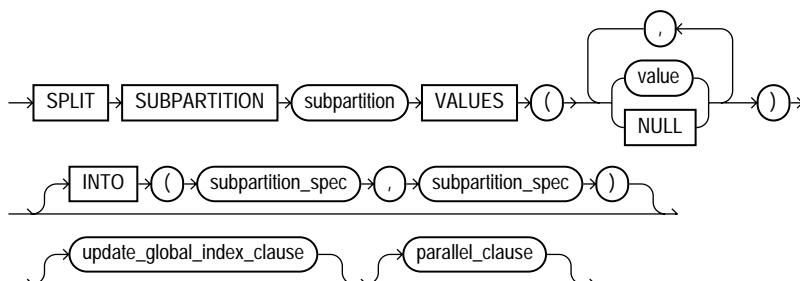
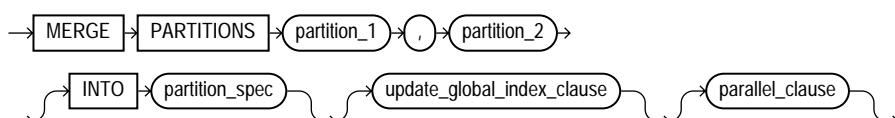


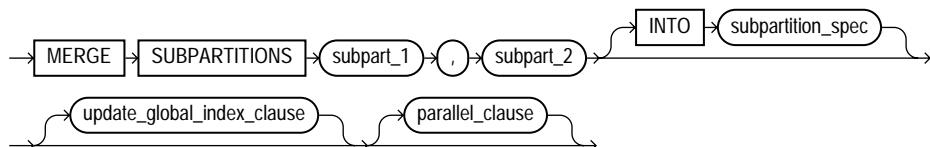
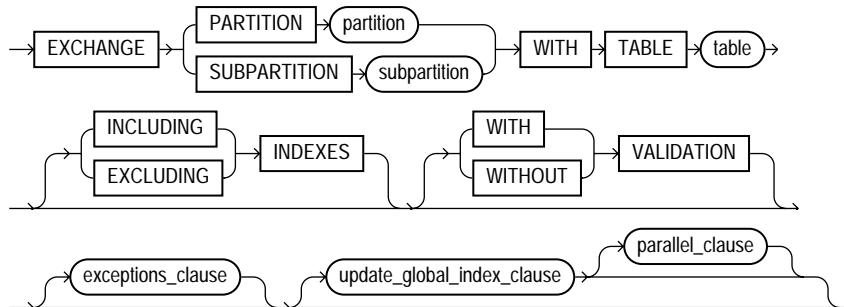
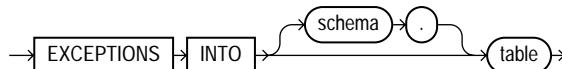
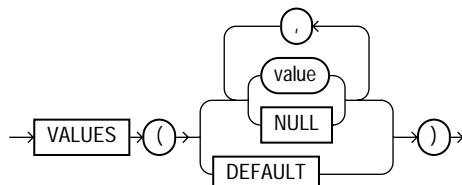
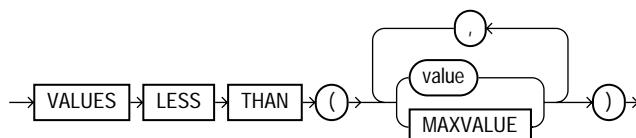
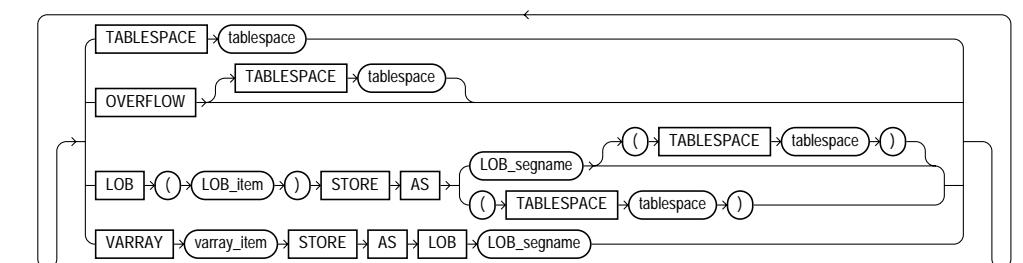
**coalesce\_table\_partition**

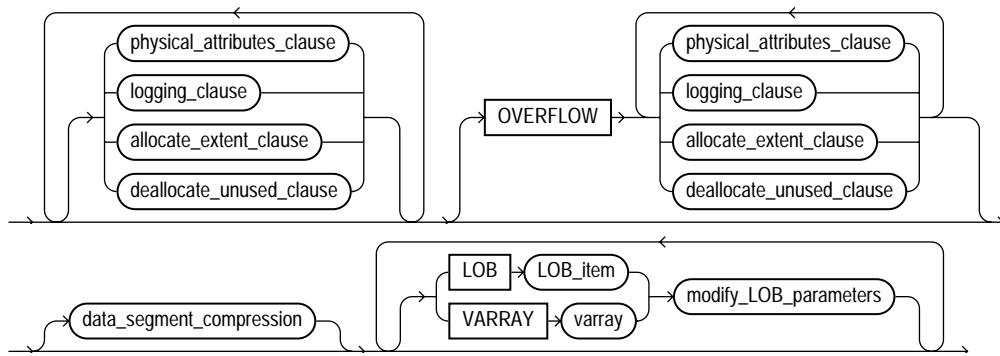
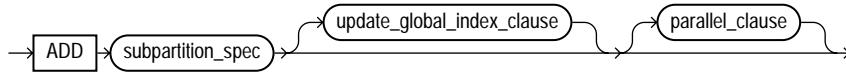
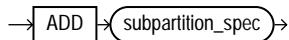
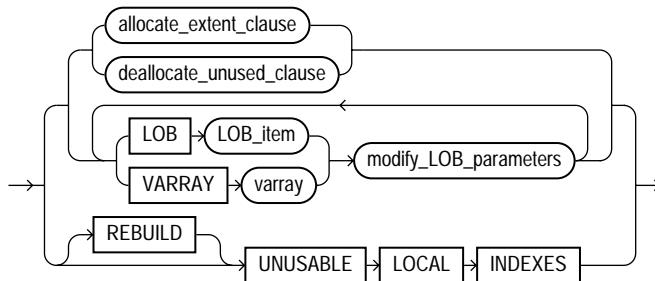
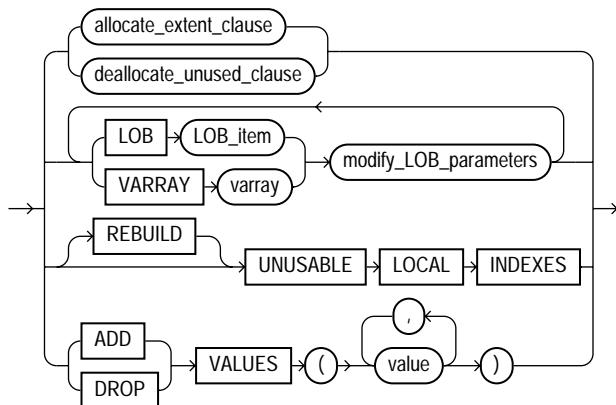


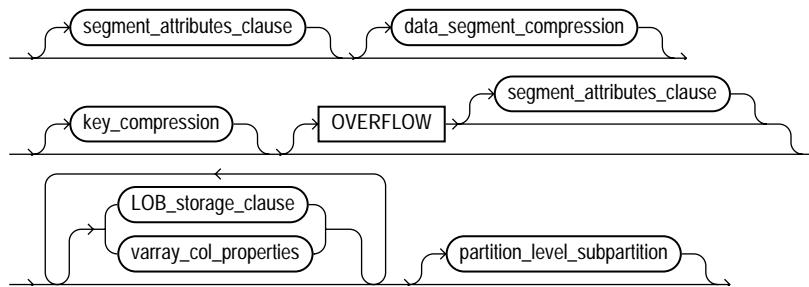
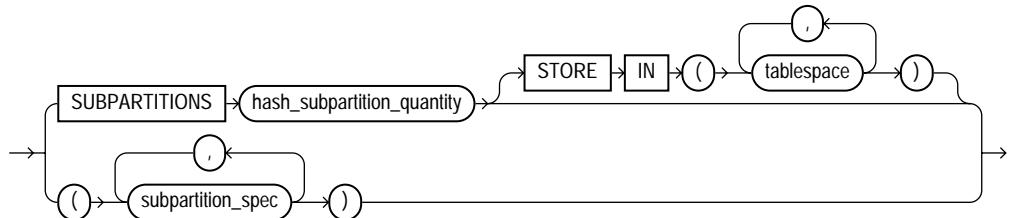
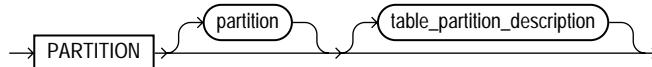
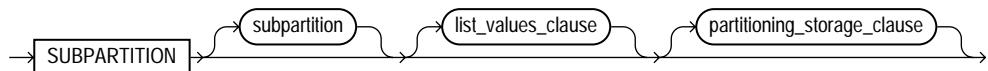
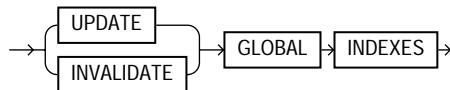
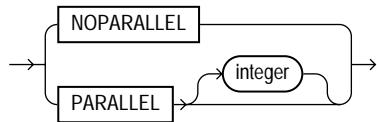
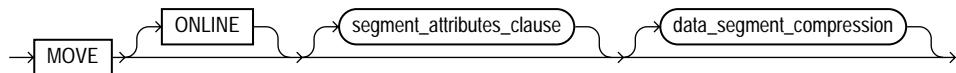
**drop\_table\_partition**

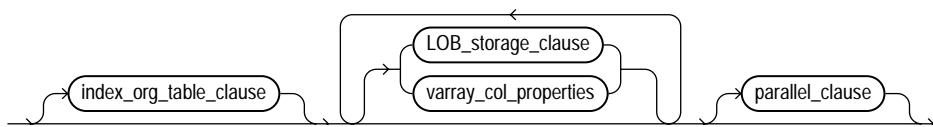
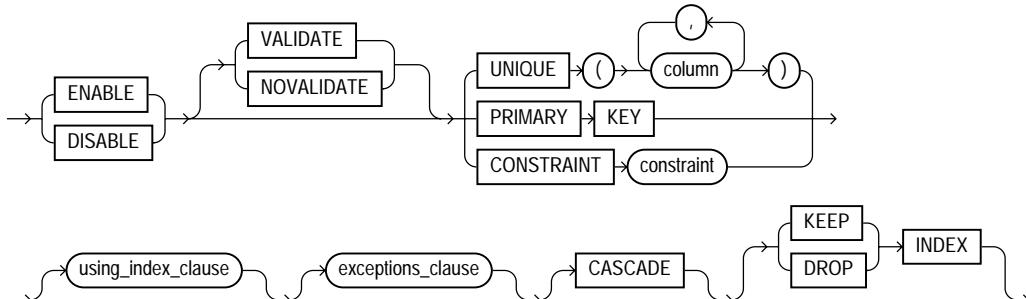
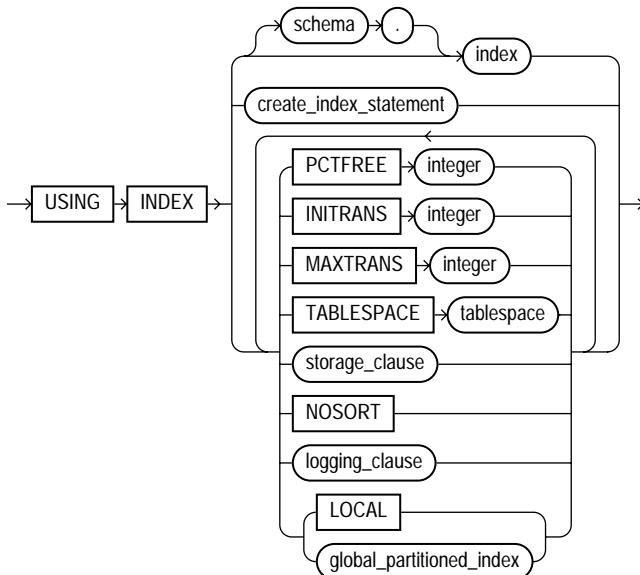
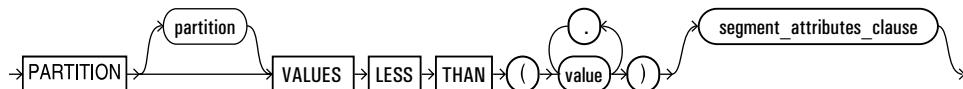


**drop\_table\_subpartition****rename\_partition\_subpart****truncate\_partition\_subpart****split\_table\_partition****split\_table\_subpartition****merge\_table\_partitions**

**merge\_table\_subpartitions****exchange\_partition\_subpart****exceptions\_clause****list\_values\_clause****range\_values\_clause****partitioning\_storage\_clause**

**partition\_attributes****add\_hash\_subpartition****add\_list\_subpartition****modify\_hash\_subpartition****modify\_list\_subpartition**

**table\_partition\_description****partition\_level\_subpartition****partition\_spec****subpartition\_spec****update\_global\_index\_clause****parallel\_clause****move\_table\_clause**

**enable\_disable\_clause****using\_index\_clause****global\_partitioned\_index****index\_partitioning\_clause**

**DESCRIZIONE** ALTER TABLE modifica la definizione di una tabella esistente. ADD consente di aggiungere una nuova colonna alla fine di una tabella esistente oppure aggiungere una limitazione alla definizione della tabella. Viene utilizzato lo stesso formato di CREATE TABLE.

MODIFY modifica una colonna esistente, con alcune restrizioni.

- È possibile modificare il tipo di colonna o ridurne la dimensione solo se ciascuna riga della colonna è NULL.
- Una colonna NOT NULL può essere aggiunta solo a una tabella che non possiede alcuna riga.
- Una colonna esistente può essere modificata in NOT NULL solo se possiede un valore non nullo per ciascuna riga o se viene specificato un valore predefinito durante la modifica.
- Incrementando la lunghezza di una colonna NOT NULL senza specificare NULL, questa viene lasciata invariata a NOT NULL.

È possibile utilizzare la clausola SET UNUSED per contrassegnare le colonne come inutilizzate. Quando si utilizza la clausola DROP UNUSED COLUMNS, la tabella sarà riorganizzata e tutte le colonne inutilizzate saranno rimosse.

ALLOCATE EXTENT consente di allocare in modo esplicito un nuovo extent. ENABLE e DISABLE abilitano e disabilitano i vincoli. Tutte le altre funzionalità di ALTER TABLE funzionano allo stesso modo che nel comando CREATE TABLE, tranne per il fatto che vengono applicate a una tabella esistente. Vedere CREATE TABLE per ulteriori dettagli.

Per modificare una tabella, è necessario essere in possesso del privilegio ALTER per quella tabella o del privilegio di sistema ALTER ANY TABLE.

CACHE specifica che i blocchi per questa tabella, quando letti, devono essere contrassegnati come “blocchi utilizzati più di recente” e mantenuti nella buffer cache dei blocchi di dati il più a lungo possibile. Questa opzione è utile se le tabelle sono di piccole dimensioni e abbastanza statiche. NOCACHE (l’opzione predefinita) inverte la tabella ritornando al funzionamento normale per l’elenco LRU.

PARALLEL, assieme a DEGREE e INSTANCES, specifica le caratteristiche del funzionamento in parallelo della tabella (per i database che utilizzano l’opzione Real Application Clusters). DEGREE specifica il numero di server da utilizzare per le query; INSTANCES specifica il modo in cui la tabella dev’essere suddivisa tra le istanze di Real Application Clusters per l’elaborazione in parallelo delle query. Un intero *n* indica che la tabella dev’essere suddivisa tra il numero specificato di istanze disponibili.

È possibile utilizzare ALTER TABLE per eseguire sulle partizioni i comandi ADD, DROP, EXCHANGE, MODIFY, MOVE, SPLIT o TRUNCATE. Vedere il Capitolo 18 e CREATE TABLE.

Le clausole di memorizzazione LOB specificano l’area di memorizzazione da utilizzare per i dati LOB che non saranno memorizzati nella tabella stessa (fuori linea).

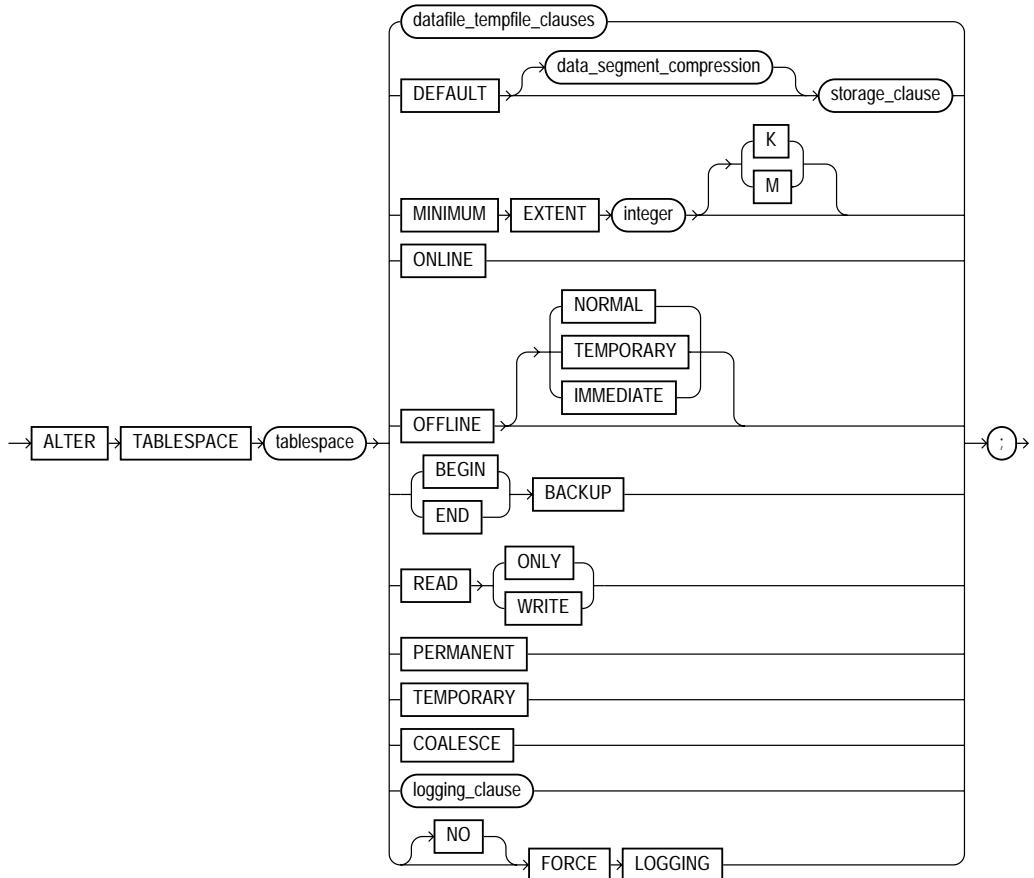
L’opzione MOVE ONLINE consente di accedere a una tabella per effettuare operazioni DML durante una riorganizzazione di tabella in linea. Le opzioni MOVE e MOVE ONLINE sono disponibili solo per le tabelle organizzate con indici e non partizionate. Durante un’operazione MOVE ONLINE, le operazioni DML parallele sulla tabella non sono supportate.

## ALTER TABLESPACE

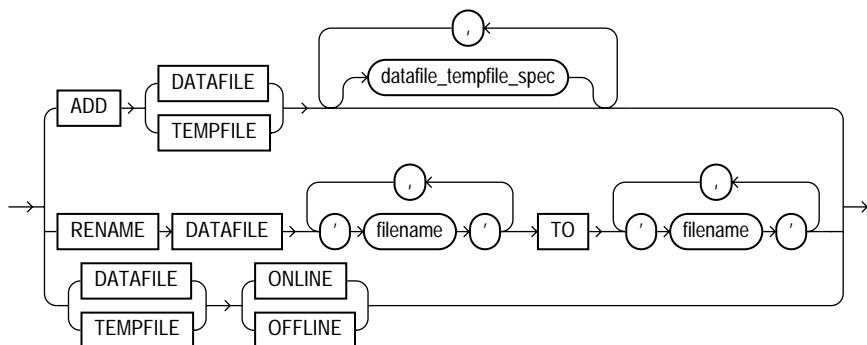
**VEDERE ANCHE** CREATE TABLESPACE, DROP TABLESPACE, STORAGE, Capitoli 20 e 40.

### SINTASSI

alter\_tablespace



**datafile\_tempfile\_clauses**



**DESCRIZIONE** *tablespace* è il nome dello spazio occupato da una tabella. ADD DATAFILE aggiunge alla tablespace un file o una serie di file descritti in base a una *definizione\_file*, che specifica i nomi e le dimensioni dei file del database: il formato del nome del file è specifico del sistema operativo. SIZE rappresenta il numero di byte riservati per questo file. Specificando il suffisso K, il valore viene moltiplicato per 1024; specificando M viene moltiplicato per 1048576. REUSE (senza SIZE) significa distruggere il contenuto di qualunque file con quel nome e assegnare il nome in modo che sia usato da questa tablespace. SIZE e REUSE creano il file, se non esiste, e ne controllano la dimensione, nel caso esista già. SIZE da solo crea il file se non esiste, ma in caso contrario restituisce un errore.

È possibile utilizzare la clausola AUTOEXTEND per ridimensionare dinamicamente i propri file di dati, se necessario, incrementandoli di una dimensione pari a NEXT, fino a un massimo di MAXSIZE (oppure UNLIMITED).

NOLOGGING specifica l'azione predefinita per la scrittura di voci di redo log per determinati tipi di transazioni nella tablespace. È possibile ridefinire questa impostazione a livello di oggetto. Vedere il Capitolo 18.

RENAME cambia il nome del file contenente una tablespace esistente. La tablespace dev'essere sconnessa al momento in cui avviene il cambio del nome. Si osservi che RENAME in realtà non rinomina i file; associa solamente i loro nuovi nomi con la tablespace in questione. La modifica vera e propria dei nomi dei file all'interno del sistema operativo dev'essere effettuata all'interno del sistema operativo stesso. Per cambiare in modo appropriato i nomi ai file, per prima cosa occorre eseguire il comando ALTER TABLESPACE OFFLINE, rinominare i file all'interno del sistema operativo, eseguendo un RENAME dei file con ALTER TABLESPACE e, successivamente, eseguire il comando ALTER TABLESPACE ONLINE.

DEFAULT STORAGE definisce lo spazio di memorizzazione predefinito per tutti i futuri oggetti creati nella tablespace in questione, fatta eccezione per quei valori predefiniti che vengono reimpostati, per esempio tramite un'operazione CREATE TABLE. ONLINE riporta in linea la tablespace. OFFLINE la disconnette, senza aspettare che i suoi utenti si scollighino (opzione IMMEDIATE) oppure dopo che tutti gli utenti hanno terminato la sessione (opzione NORMAL).

MINIMUM EXTENT imposta la dimensione minima per qualunque extent creato nella tablespace. COALESCE combina gli extent liberi vicini tra loro nella tablespace in un numero minore di extent liberi più grandi.

Le tablespace READ ONLY non vengono mai aggiornate da Oracle e possono essere collocate su supporti di sola lettura. Le tablespace di sola lettura non hanno bisogno di backup ripetuti. Tutti le tablespace vengono create con la possibilità di lettura e scrittura. Per riportare una tablespace di sola lettura nello stato lettura-scrittura, si utilizza la clausola READ WRITE.

BEGIN BACKUP può essere eseguito in qualsiasi momento. Esso colloca la tablespace in uno stato tale che un backup in linea può avvenire mentre vi sono utenti che accedono alla tablespace. END BACKUP indica che il backup del sistema è finito. Se la tablespace è in linea, qualunque salvataggio del sistema deve salvare anche i redo log dell'archivio. Se invece non è in linea, questa operazione non è necessaria.

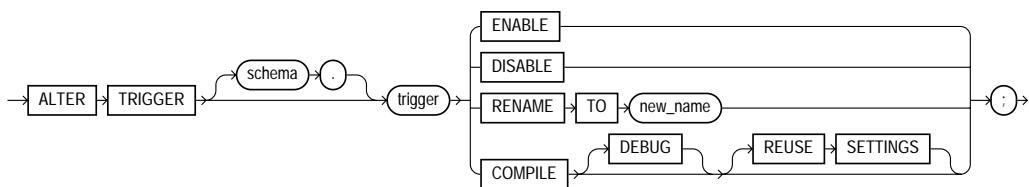
Le tablespace TEMPORARY non possono contenere alcun oggetto permanente (quali tabelle o indici); possono solamente contenere i segmenti temporanei utilizzati da Oracle durante le operazioni di ordinamento e di creazione di indici. Una tablespace PERMANENT può contenere qualsiasi tipo di segmento di Oracle.

## ALTER TRIGGER

**VEDERE ANCHE** CREATE TRIGGER, DROP TRIGGER, ENABLE, TRIGGER, Capitolo 28.

## SINTASSI

`alter_trigger`



**DESCRIZIONE** ALTER TRIGGER abilita, disabilita, rinomina o ricompila un trigger PL/SQL. Vedere CREATE TRIGGER e il Capitolo 28 per una trattazione riguardante i trigger. Per eseguire il comando ALTER TRIGGER è necessario possedere il trigger oppure disporre del privilegio di sistema ALTER ANY TRIGGER.

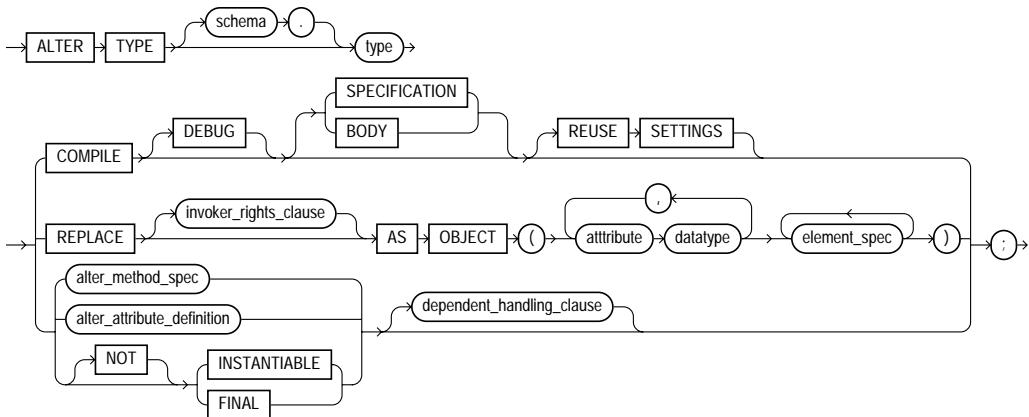
Per ricompilare manualmente un trigger non corretto si utilizza la clausola COMPILE. Dal momento che i trigger dipendono da altri elementi, possono divenire non validi se un oggetto su cui si basa il trigger viene modificato. La clausola DEBUG consente la generazione di informazioni PL/SQL durante la ricompilazione del trigger.

## ALTER TYPE

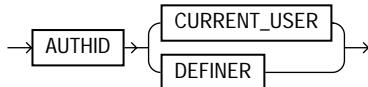
**VEDERE ANCHE** CREATE TYPE, CREATE TYPE BODY, Capitoli 4 e 30.

### SINTASSI

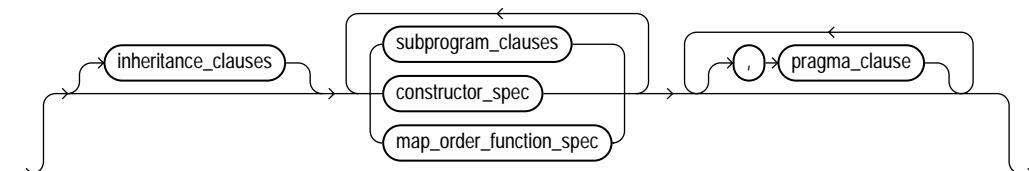
`alter_type`

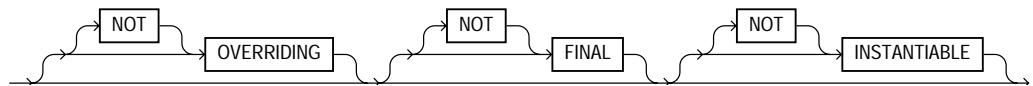
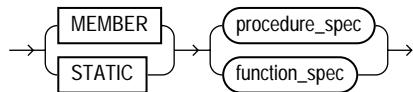
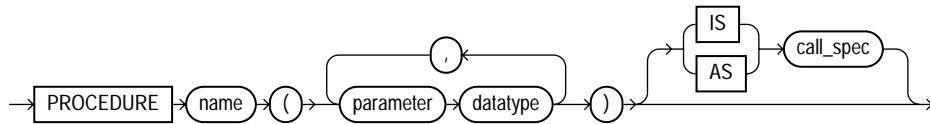
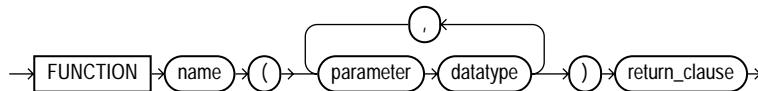
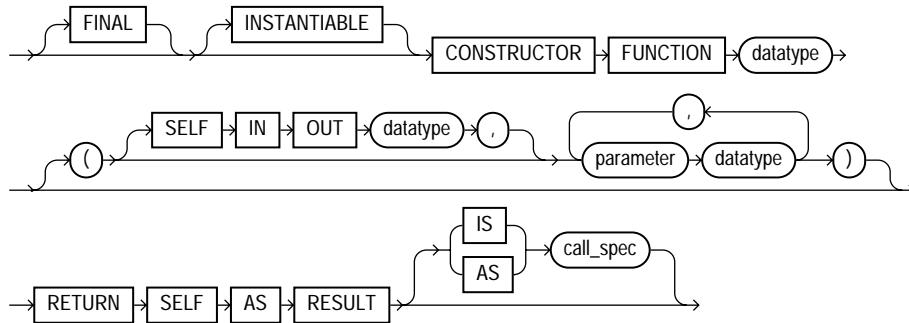
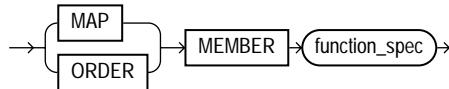
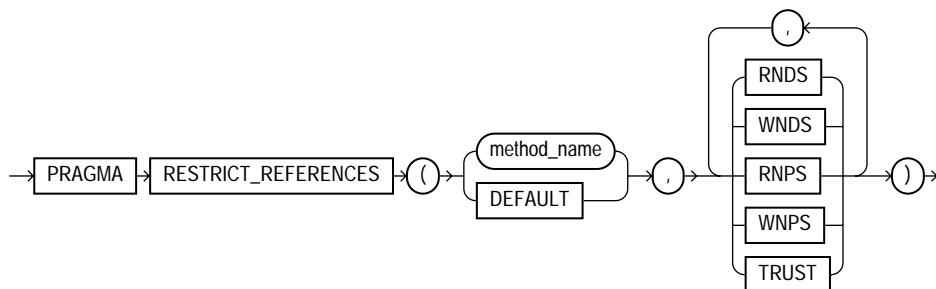


`Invoker_rights_clause`

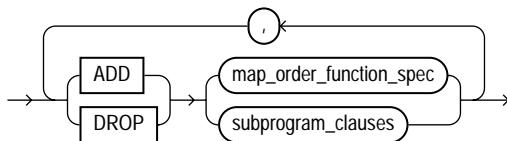


`element_spec`

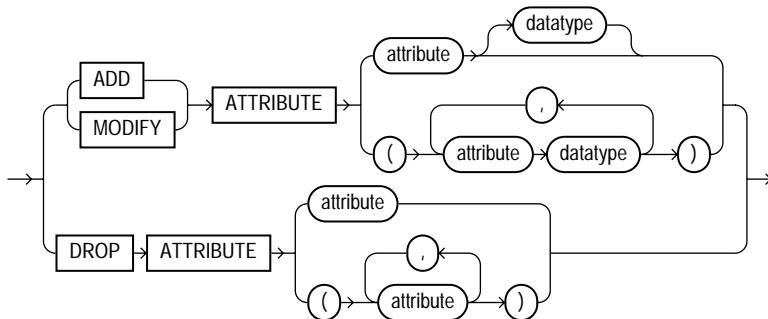


**inheritance\_clauses****subprogram\_clauses****procedure\_spec****function\_spec****constructor\_spec****map\_order\_function\_spec****pragma\_clause**

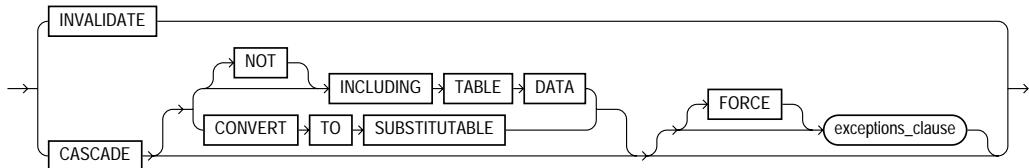
## alter\_method\_spec



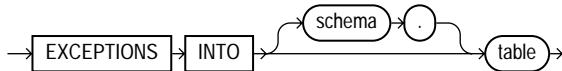
## alter\_attribute\_definition



## dependent\_handling\_clause



## exceptions\_clause



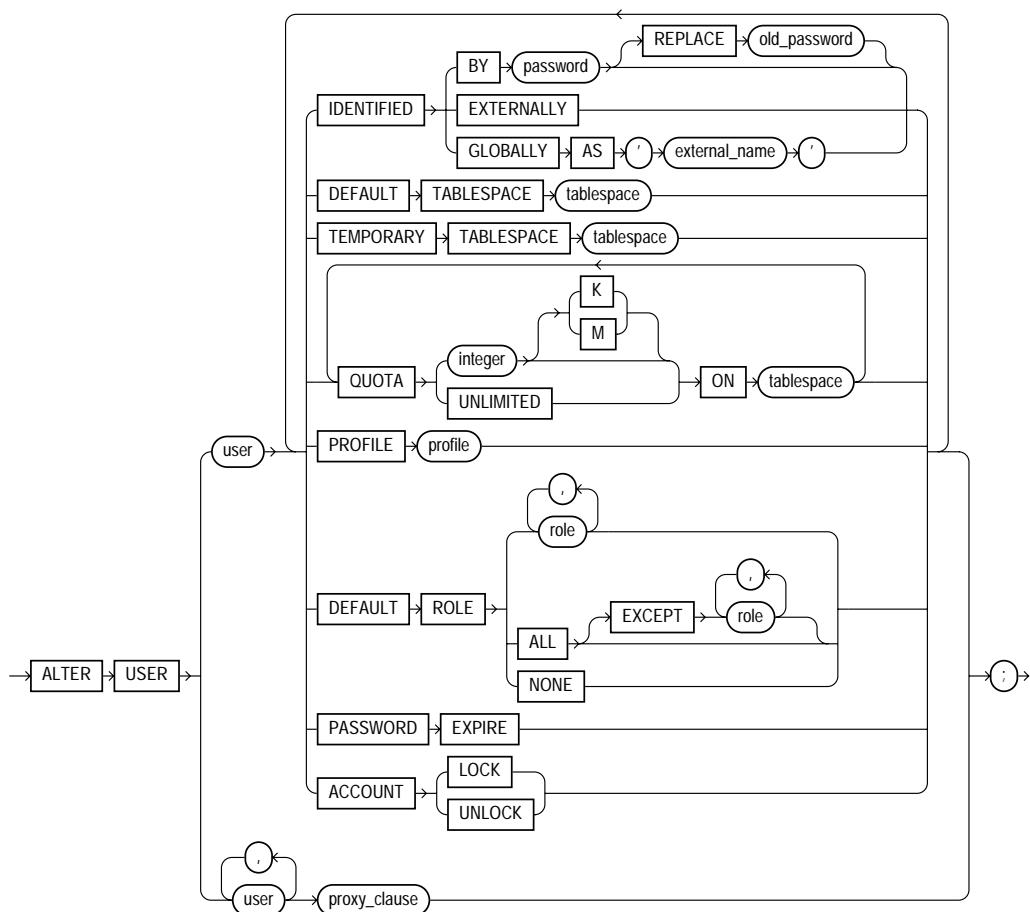
**DESCRIZIONE** ALTER TYPE consente di modificare i tipi di dati astratti esistenti. Quando viene creato un metodo che agisce sul tipo di dato astratto, si utilizza il comando CREATE TYPE BODY (*vedere* la voce corrispondente in questo capitolo). Ciascun metodo definito nel corpo del tipo deve prima essere elencato all'interno del tipo. Le funzioni membro normalmente utilizzano l'opzione PRAGMA RESTRICT\_REFERENCES con il vincolo WNDS (Write No Database State).

**ALTER USER**

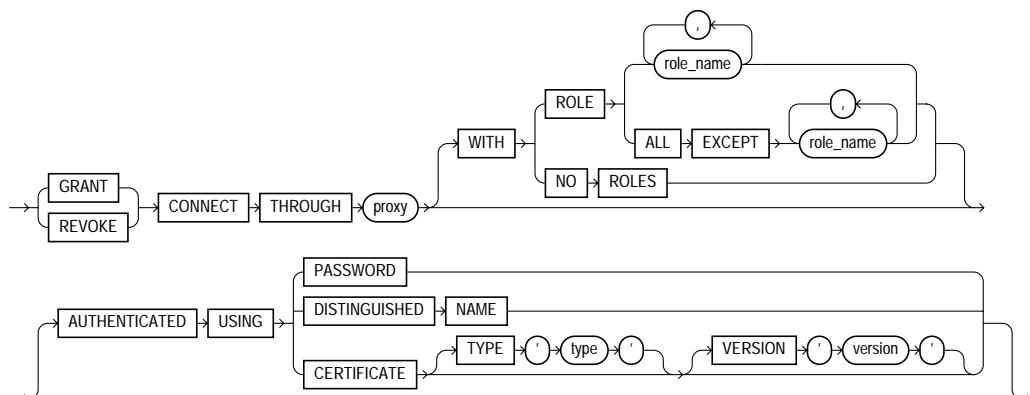
**VEDERE ANCHE** CREATE TABLESPACE, GRANT, CREATE PROFILE, CREATE USER, Capitolo 19.

**SINTASSI**

alter\_user



proxy\_clause



**DESCRIZIONE** Si utilizza ALTER USER per modificare la password di un utente, la tablespace DEFAULT (per gli oggetti di proprietà dell'utente) o le tablespace TEMPORARY (per segmenti temporanei utilizzati dall'utente). Senza ALTER USER, i valori predefiniti per entrambi vengono impostati secondo i valori predefiniti della prima tablespace (sia per gli oggetti sia per i segmenti temporanei) concessa all'utente come risorsa tramite GRANT e della tablespace temporanea predefinita a livello del database. ALTER USER consente inoltre di modificare la quota, il profilo della risorsa o il ruolo predefinito (*vedere CREATE USER*). È possibile utilizzare la clausola PASSWORD EXPIRE per forzare la scadenza della password di un utente, nel qual caso l'utente dovrà cambiare la propria password al successivo collegamento. La clausola ACCOUNT UNLOCK consente di sbloccare un account che ha superato il numero massimo di tentativi di collegamento consecutivi falliti. *Vedere CREATE PROFILE* per ulteriori dettagli sul controllo della password. Per modificare un utente è necessario possedere il privilegio di sistema ALTER USER.

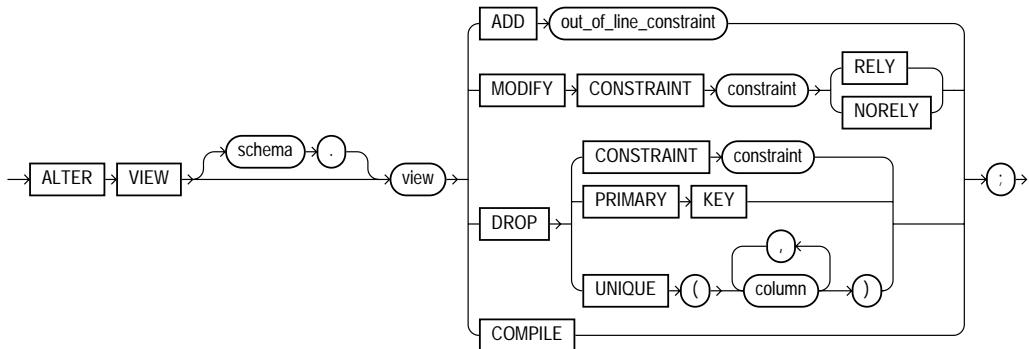
È possibile utilizzare la clausola PROXY per connettersi come l'utente specificato e attivare tutti i ruoli dell'utente, o solo qualcuno o anche nessuno. PROXY normalmente viene impiegato nelle applicazioni a tre livelli che implicano l'uso di server applicativi.

## ALTER VIEW

**VEDERE ANCHE** CREATE VIEW, DROP VIEW, Capitolo 18.

### SINTASSI

alter\_view



**DESCRIZIONE** ALTER VIEW ricompila una vista o ne modifica i vincoli. Per modificare una vista è necessario esserne il proprietario oppure disporre del privilegio di sistema ALTER ANY TABLE.

## ALTRÉ FUNZIONI

Di seguito è riportato un elenco alfabetico di tutte le funzioni SQL di Oracle che non possono essere classificate all'interno delle altre categorie. Ciascuna viene riportata altrove in questa guida, assieme alla sintassi e ad esempi d'uso.

DUMP( *stringa* [, *formato* [, *inizio* [, *lunghezza*] ] ] )

DUMP visualizza il valore di *stringa* in un formato di data interno, in ASCII, in formato ottale, decimale, esadecimale o di carattere.

**NVL(*valore*, *sostituto*)**

Se *valore* è NULL, la funzione NVL restituisce *sostituto*. Altrimenti restituisce *valore*.

**NVL2 ( *espr1* , *espr2* , *espr3* )**

Se *espr1* non è NULL, NVL2 restituisce *espr2*. Se *espr1* è NULL, NVL2 restituisce *espr3*.

**VSIZE(*espressione*)**

VSIZE indica il numero di byte di cui Oracle ha bisogno per memorizzare *espressione* nel database.

## ANALISI

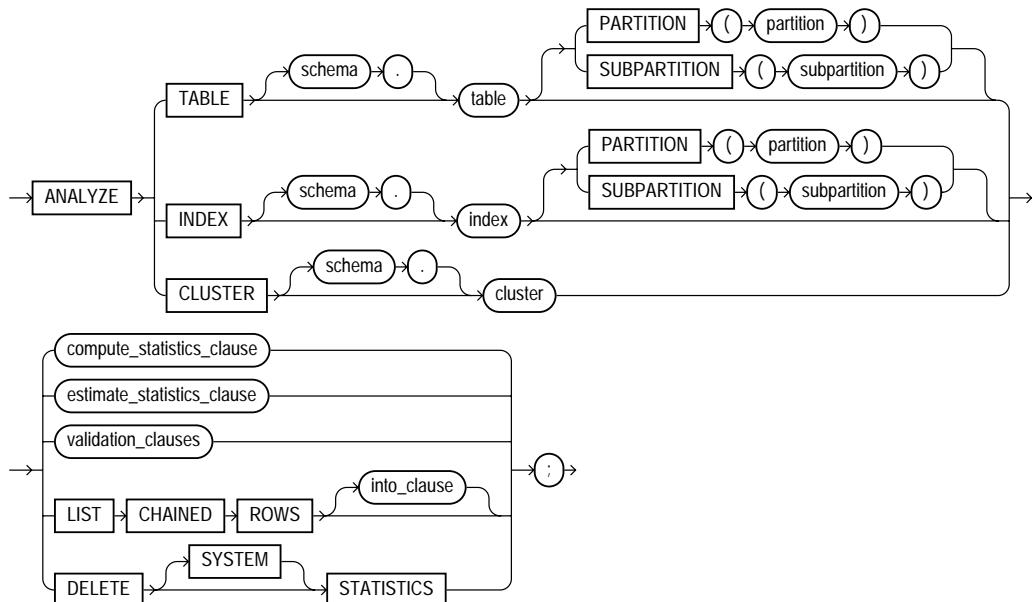
L'analisi è la corrispondenza di un'istruzione SQL a un cursore. A questo stadio, vengono effettuati diversi controlli di convalida quali la verifica di esistenza di tutti gli oggetti indicati, il controllo della correttezza delle concessioni e il controllo sintattico e semantico e così via. Inoltre, vengono prese decisioni riguardanti l'esecuzione e l'ottimizzazione del codice come la scelta degli indici e così via.

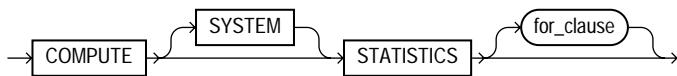
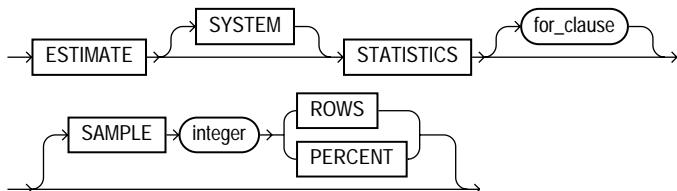
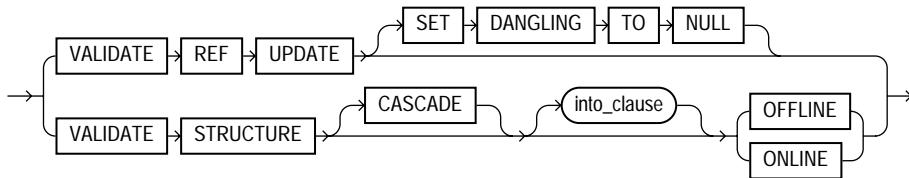
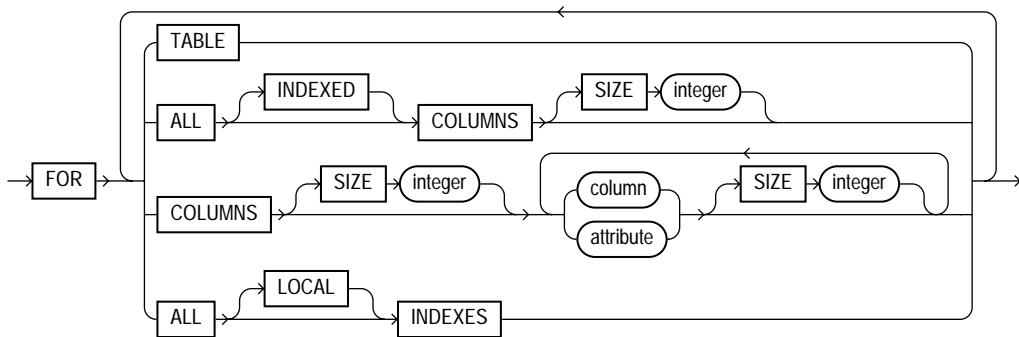
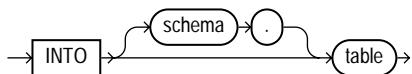
## ANALYZE

**VEDERE ANCHE** Capitolo 38.

### SINTASSI

**analyze**



**compute\_statistics\_clause****estimate\_statistics\_clause****validation\_clauses****for\_clause****into\_clause**

**DESCRIZIONE** Oracle consiglia di utilizzare il package DBMS\_STATS al posto del comando ANALYZE, tuttavia quest'ultimo offre capacità non ancora disponibili in DBMS\_STATS. ANALYZE, a differenza di DBMS\_STATS, supporta la capacità di campionare un numero specifico di righe, di convalidare un indice, di elencare le righe concatenate e di raccogliere dati in blocchi a elenco libero.

ANALYZE consente di riunire per l'ottimizzatore informazioni statistiche relative a una tabella, un cluster, una partizione o un indice, memorizzandole in un dizionario di dati; consente di cancellare queste informazioni, di convalidare la struttura di un oggetto e di identificare le righe di una tabella o di un cluster che sono state spostate o che sono state concatenate, con un elenco

in una tabella locale. La valutazione di questi elementi è solitamente più veloce e presenta la stessa precisione di quella elaborata dal comando COMPUTE che fornisce le informazioni statistiche dell'ottimizzatore. Per controllare se un oggetto presenta una possibile corruzione dei dati, è possibile utilizzare la clausola VALIDATE STRUCTURE. Per analizzare una tabella, è necessario esserne il proprietario o possedere il privilegio di sistema ANALYZE ANY.

Ai comandi COMPUTE STATISTICS ed ESTIMATE STATISTICS può essere associata la clausola FOR. La clausola FOR consente di modificare il funzionamento predefinito di ANALYZE. È possibile analizzare una tabella senza i relativi indici (FOR TABLE), una tabella e tutte le sue colonne indicizzate (FOR TABLE FOR INDEXED COLUMNS) o solamente alcune colonne. Il parametro SIZE serve a riempire istogrammi di dati utilizzati dall'ottimizzatore basato sui costi durante le operazioni di messa a punto di tipo avanzato.

La clausola LIST CHAINED ROWS registra informazioni relative alle righe concatenate in una tabella specificata.

## **AND**

Vedere OPERATORI LOGICI, PRECEDENZE, CONTAINS.

## **ANNIDAMENTO**

L'annidamento è l'atto di inglobare un'istruzione, una clausola, una query e così via all'interno di un'altra istruzione, clausola, query e così via.

## **ANSI**

L'American National Standards Institute stabilisce una serie di standard per il linguaggio SQL e per molti dei suoi elementi.

## **ANY**

**VEDERE ANCHE** ALL, BETWEEN, EXISTS, IN, OPERATORI LOGICI, Capitolo 12.

### **SINTASSI**

*operatore ANY elenco*

**DESCRIZIONE** = ANY è l'equivalente di IN. operatore rappresenta uno dei seguenti operatori =, >, >=, <, <=, != ed *elenco* può essere una serie di stringhe letterali (quali 'Mario', 'Giovanni' o 'Ilaria') o di numeri letterali (quali 2, 43, 76, 32.06 o 444) oppure una colonna ricavata da una subquery, in cui ciascuna riga della subquery diventa membro di un elenco, come mostrato di seguito:

LOCALITA.Citta = ANY (select Citta from METEO)

Può inoltre rappresentare una serie di colonne nella clausola where della query principale, come mostrato di seguito:

Prospetto = ANY (Venditore, Cliente)

## LIMITAZIONI

*elenco* non può essere una serie di colonne di una subquery del tipo:

```
Prospetto = ANY (select Venditore, Cliente from . . .)
```

Molti trovano difficoltà nel ricordare questo operatore, come per l'operatore ALL, perché la logica di alcuni casi non è immediatamente intuitiva. Di conseguenza vengono solitamente sostituiti con alcune forme di EXISTS.

La combinazione di un operatore con ANY e con un elenco può risultare più chiara grazie alla tabella seguente:

Pag = ANY (4,2,7)	Pag è nell'elenco; sia 2 che 4 e 7 soddisfano la condizione.
Pag > ANY (4,2,7)	Pag è maggiore di qualsiasi singolo elemento nell'elenco (4,2,7); anche 3 soddisfa la condizione, perché è maggiore di 2.
Pag >= ANY (4,2,7)	Pag è maggiore o uguale a qualsiasi elemento nell'elenco (4,2,7); anche 2 soddisfa la condizione, perché è uguale a 2.
Pag < ANY (4,2,7)	Pag è minore di qualsiasi elemento dell'elenco (4,2,7); anche 6 soddisfa la condizione perché è minore di 7.
Pag <= ANY (4,2,7)	Pag è minore o uguale a qualsiasi elemento dell'elenco (4,2,7); anche 7 soddisfa la condizione.
Pag != ANY (4,2,7)	Pag è diverso da qualsiasi elemento singolo dell'elenco; qualsiasi numero soddisfa la condizione poiché l'elenco contiene più di un valore. Con un solo valore, !=ANY equivale a !=.

## APPEND

**VEDERE ANCHE** CHANGE, DEL, EDIT, LIST, Capitolo 6.

### SINTASSI

```
A[PPEND] testo
```

**DESCRIZIONE** APPEND è una funzionalità dell'editor a linea di comando di SQL\*PLUS. Questo comando posiziona il testo alla fine della riga corrente nel buffer corrente, senza interporre spazi. Se si desidera inserire uno spazio tra la fine della riga corrente e il testo, è sufficiente inserire due caratteri di spaziatura tra il comando APPEND e il testo. Analogamente, se si desidera scrivere un punto e virgola alla fine della riga, occorre specificarne due consecutivi (uno di essi viene considerato come simbolo di fine comando e scartato).

### ESEMPIO

```
APPEND ::
```

## APPLICAZIONE

Un'applicazione rappresenta un insieme di moduli, menu, report e altri componenti in grado di soddisfare una funzione commerciale particolare. Per esempio, vi sono applicazioni che servono come sistemi per l'acquisizione di ordini di merci.

## ARCH, PROCESSO

Uno dei processi in background utilizzati da Oracle; ARCH esegue archiviazioni automatiche di file redo log quando il database viene utilizzato in modalità ARCHIVELOG. Vedere BACKGROUND, PROCESSO.

## ARCHIVE LOG

**VEDERE ANCHE** ALTER DATABASE, RECOVER, Capitolo 40.

### SINTASSI

```
ARCHIVE LOG {LIST|STOP}|{START|NEXT|ALL|intero} [TO destinazione]
```

**DESCRIZIONE** ARCHIVE LOG avvia o interrompe l'archiviazione automatica dei file di redo log in linea, archivia manualmente file di redo log specifici, oppure visualizza informazioni sui file di redo log. START abilita l'archiviazione automatica, mentre STOP la disabilita. LIST mostra lo stato di archiviazione dei redo log in linea. NEXT archivia manualmente il gruppo di file di redo log successivo che è stato riempito ma non ancora archiviato. ALL archivia manualmente tutti i gruppi di file di redo log riempiti ma non ancora archiviati. È possibile specificare il numero del gruppo di file di redo log in linea e l'area di destinazione per i file archiviati.

### ESEMPIO

```
ARCHIVE LOG ALL  
ARCHIVE LOG LIST
```

## ARCHIVIARE

In generale, archiviare significa salvare i dati per un possibile utilizzo futuro. In senso specifico, significa salvare i dati trovati nei redo log in linea, nel caso che i log risultino necessari per ripristinare il database in seguito a un errore verificatosi in uno dei supporti.

## AREA DI CONTESTO

Un'area di contesto è un'area di lavoro all'interno della memoria in cui Oracle memorizza l'istruzione SQL corrente e, se si tratta di una query, una riga di risultato. L'area di contesto contiene lo stato di un cursore.

## ARGOMENTO

Un argomento è un'espressione all'interno delle parentesi di una funzione o procedura che fornisce un valore che la funzione o procedura può utilizzare.

## ARRAY, ELABORAZIONE

L'elaborazione di un array viene eseguita su insiemi di dati anziché su una riga per volta. In alcune utility di Oracle, quali Export/Import e i precompilatori, gli utenti possono impostare la dimensione dell'array; generalmente, aumentando la dimensione, si migliorano le prestazioni.

## ARRAY VARIABILE

Un array variabile è un collettore. Per ciascuna riga di una tabella è in grado di contenere più voci. Il numero massimo di dati di un array variabile per riga viene impostato alla sua creazione (con CREATE TYPE). Vedere il Capitolo 31.

## ARRAYSIZE (SQL\*PLUS)

Vedere SET.

## AS

**VEDERE ANCHE** ALIAS, TO\_CHAR, Capitolo 7 e Capitolo 9.

**DESCRIZIONE** AS viene utilizzato per separare le formule delle colonne dagli alias delle colonne.

### ESEMPIO

In questo esempio AS separa l'alias GiornoPaga della colonna dalla formula della colonna stessa:

```
select GIORNO_SUCC(CicloData,'VENERDI') AS GiornoPaga  
from GIORNOPAGA;
```

## ASCII

**VEDERE ANCHE** CARATTERI, FUNZIONI; ASCIISTR; CHR.

### SINTASSI

ASCII(*stringa*)

**DESCRIZIONE** ASCII è l'acronimo di “American Standard Code for Information Interchange”, una convenzione che consente di utilizzare dati digitali per rappresentare caratteri da stampare.

La funzione ASCII restituisce il valore ASCII del primo carattere (quello più a sinistra) nella stringa. Il valore ASCII di un carattere è un intero compreso tra 0 e 255. Quelli tra 0 e 127 hanno uno standard ben definito. Quelli superiori al 127 (“set ASCII esteso”) differiscono da Paese a Paese, da applicazione ad applicazione e in relazione al costruttore del computer. La lettera A, per esempio, è uguale al numero ASCII 65, B è 66, C è 67 e così via. Il punto decimale è 46. Il segno meno è 45. Il numero 0 è 48, 1 è 49, 2 è 50 e così via.

### ESEMPIO

```
select ASCII('.'), ASCII(.5),  
       ASCII('E'), ASCII('EMILY')  
  from DUAL;  
  
----- ----- -----  
 ASCII('.') ASCII(.5) ASCII('E') ASCII('EMILY')  
----- ----- -----  
 46        46        69        69
```

## ASCIISTR

**VEDERE ANCHE** CARATTERI, FUNZIONI; ASCII; CHR.

### SINTASSI

ASCIISTR(*stringa*)

**DESCRIZIONE** La funzione ASCII prende una stringa e restituisce il valore ASCII equivalente nel set di caratteri del database.

### ESEMPIO

```
select ASCISTR('E'), ASCISTR('EMILY')
  from DUAL;
```

A ASCII

- -----

E EMILY

## ASIN

**VEDERE ANCHE** ACOS, ATAN, ATAN2, COS, COSH, EXP, LN, LOG, SIN, SINH, TAN, TANH.

### SINTASSI

ASIN(*valore*)

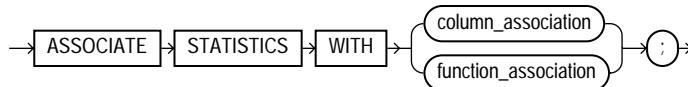
**DESCRIZIONE** ASIN restituisce l'arcoseno di un valore. I valori di input variano da -1 a 1; gli output sono espressi in radianti.

## ASSOCIATE STATISTICS

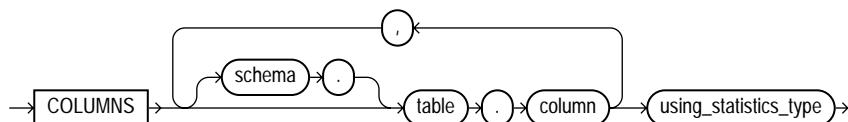
**VEDERE ANCHE** ANALYZE.

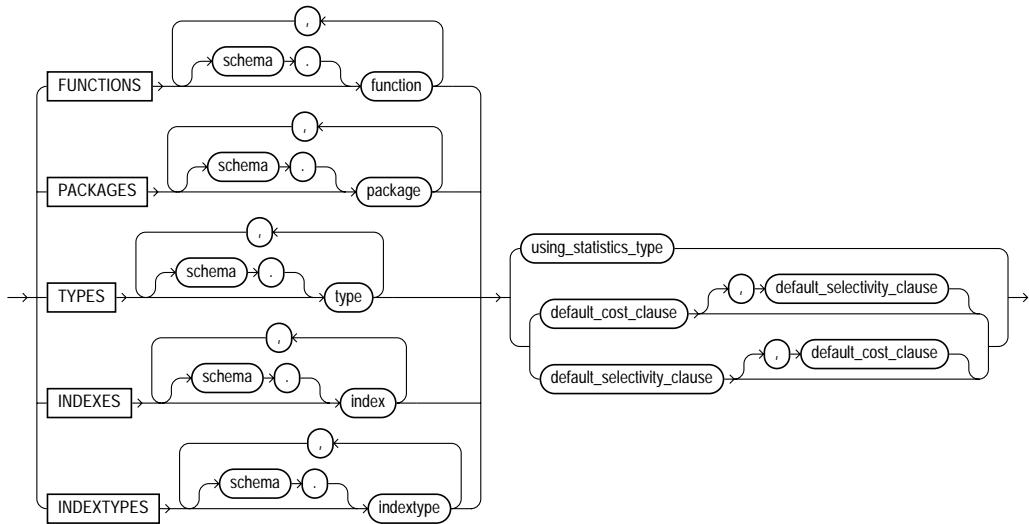
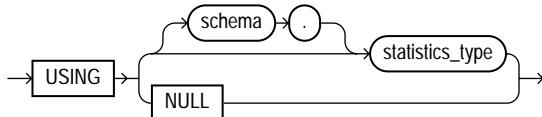
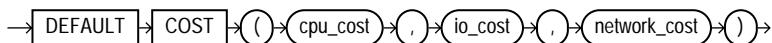
### SINTASSI

associate\_statistics



### column\_association



**function\_association****using\_statistics\_type****default\_cost\_clause****default\_selectivity\_clause**

**DESCRIZIONE ASSOCIATE STATISTICS** associa un set di funzioni statistiche con una o più colonne, funzioni standalone, package, tipi o indici. È necessario disporre dei privilegi richiesti per poter applicare il comando ALTER all'oggetto modificato.

È possibile vedere le associazioni in USER\_USTATS dopo aver analizzato l'oggetto con il quale si stanno associando le proprie funzioni statistiche.

## ASTRATTI, TIPI DI DATI

I *tipi di dati astratti* sono tipi di dati costituiti da uno o più attributi. Invece di avere i vincoli dei tipi di dati standard NUMBER, DATE e VARCHAR2, i tipi di dati astratti possono essere costituiti da più attributi e/o da altri tipi di dati astratti definiti in precedenza. I tipi di dati astratti sono noti anche come *tipi definiti dall'utente* (UDT, User Defined Types). Fare riferimento ai Capitoli 4 e 30 per degli esempi e a CREATE TYPE per la sintassi di creazione.

## ATAN

**VEDERE ANCHE** ACOS, ASIN, ATAN2, COS, COSH, EXP, LN, LOG, SIN, SINH, TAN, TANH.

### SINTASSI

ATAN(*valore*)

**DESCRIZIONE** ASIN restituisce l'arcotangente di un valore. I valori di input sono infiniti; gli output sono espressi in radianti.

## ATAN2

**VEDERE ANCHE** ACOS, ASIN, ATAN, COS, COSH, EXP, LN, LOG, SIN, SINH, TAN, TANH.

### SINTASSI

ATAN2(*valore1*, *valore2*)

**DESCRIZIONE** ATAN2 restituisce l'arcotangente di due valori. I valori di input sono infiniti; gli output sono espressi in radianti.

## ATTRIBUTE

Un attributo può essere uno dei tre elementi seguenti:

- un sinonimo di “caratteristica” o “proprietà”;
- un altro nome per una colonna;
- una parte di un tipo di dato astratto.

**VEDERE ANCHE** COLONNA, Capitolo 30.

### SINTASSI

ATTRIBUTE [*nome\_tipo.nome\_attributo* [*opzione ...*]]

*Opzione* può essere uno degli elementi riportati di seguito:

```
ALI[AS] alias
CLE[AR]
FOR[MAT] sintassi
LIKE {nome_tipo.nome_attributo|alias}
ON|OFF
```

**DESCRIZIONE** ATTRIBUTE imposta gli attributi di visualizzazione per gli attributi di una colonna di tipo definito dall'utente in SQL\*PLUS.

### ESEMPIO

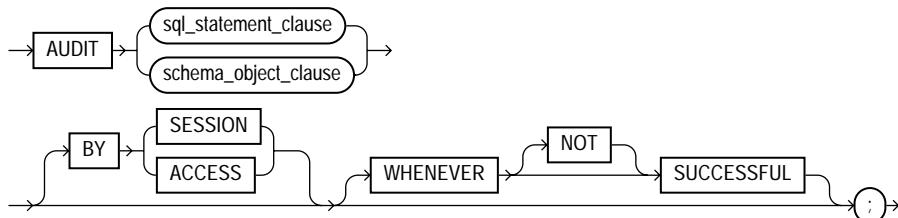
ATTRIBUTE ADDRESS\_TY.Street FORMAT A50

## AUDIT

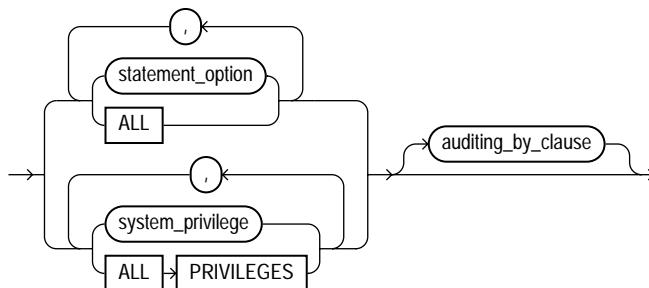
**VEDERE ANCHE** CREATE DATABASE LINK, DIZIONARIO DI DATI, NOAUDIT, PRIVILEGI.

### SINTASSI

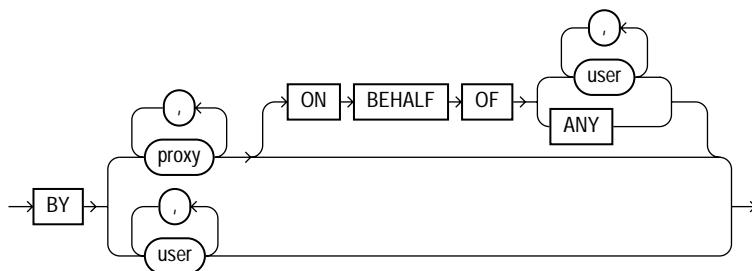
audit



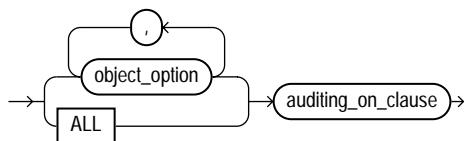
sql\_statement\_clause

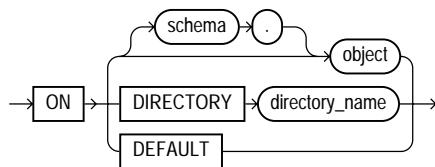


auditing\_by\_clause



schema\_object\_clause



**auditing\_on\_clause**

**DESCRIZIONE** È possibile applicare il comando AUDIT alle istruzioni eseguite dagli utenti e a quelle che accedono a oggetti specifici.

Le opzioni dell'istruzione sono le seguenti.

OPZIONE	FUNZIONE DI AUDIT
CLUSTER	Crea, modifica, elimina o tronca un cluster.
CONTEXT	Crea o elimina un contesto.
DATABASE LINK	Crea o elimina un collegamento a database.
DIMENSION	Crea, modifica o elimina una dimensione.
DIRECTORY	Crea o elimina una directory.
INDEX	Crea, modifica o elimina un indice.
NOT EXISTS	Istruzioni SQL che falliscono perché l'oggetto non esiste.
PROCEDURE	Crea o elimina una procedura, una funzione, una libreria o un package; crea il corpo del package.
PROFILE	Crea, modifica o elimina un profilo.
PUBLIC DATABASE LINK	Crea o elimina collegamenti a database pubblici.
PUBLIC SYNONYM	Crea o elimina sinonimi pubblici.
ROLE	Crea, modifica, elimina o imposta un ruolo.
ROLLBACK SEGMENT	Crea, modifica o elimina segmenti di rollback.
SEQUENCE	Crea, modifica o elimina sequenze.
SESSION	Tentativi di collegamento.
SYNONYM	Crea o elimina sinonimi.
SYSTEM AUDIT	Esegue o meno l'audit di istruzioni SQL.
SYSTEM GRANT	Esegue o meno l'audit di privilegi di sistema e ruoli.
TABLE	Crea, elimina o tronca una tabella.
TABLESPACE	Crea, modifica o elimina una tablespace.
TRIGGER	Crea, modifica o elimina un trigger; modifica una tabella con ENABLE   DISABLE ALL TRIGGERS.
TYPE	Crea, modifica o elimina un tipo o il corpo di un tipo.
USER	Crea, modifica o elimina un utente.
VIEW	Crea o modifica una vista.

ALL controlla tutte queste funzioni, ma non le seguenti.

ALTER SEQUENCE	ALTER SEQUENCE
ALTER TABLE	ALTER TABLE
COMMENT TABLE	COMMENT ON TABLE <i>tabella, vista, vista materializzata</i> COMMENT ON COLUMN <i>tabella.colonna, vista.column, vista materializzata.colonna</i>
DELETE TABLE	DELETE FROM <i>tabella, vista</i>
EXECUTE PROCEDURE	CALL Esecuzione di qualunque procedura o funzione o accesso a qualunque variabile, libreria o cursore all'interno di un package.
GRANT DIRECTORY	Concessione (GRANT) di privilegio su (ON) una directory. Revoca (REVOKE) di privilegio su (ON) una directory.
GRANT PROCEDURE	Concessione (GRANT) di privilegio su (ON) procedura, funzione, package. Revoca (REVOKE) di privilegio su (ON) procedura, funzione, package.
GRANT SEQUENCE	Concessione (GRANT) di privilegio su (ON) una sequenza. Revoca (REVOKE) di privilegio su (ON) una sequenza.
GRANT TABLE	Concessione (GRANT) di privilegio ON tabella, vista, vista materializzata. Revoca (REVOKE) di privilegio su (ON) tabella, vista, vista materializzata.
GRANT TYPE	Concessione (GRANT) di privilegio su (ON) TYPE. Revoca (REVOKE) di privilegio su (ON) TYPE.
INSERT TABLE	INSERT INTO <i>tabella, vista</i>
LOCK TABLE	LOCK TABLE <i>tabella, vista</i>
SELECT SEQUENCE	Qualunque istruzione contenente sequenza.CURRVAL o sequenza.NEXTVAL.
SELECT TABLE	SELECT FROM <i>tabella, vista, vista materializzata</i>
UPDATE TABLE	UPDATE <i>tabella, vista</i>

BY *utente* controlla solo le istruzioni SQL eseguite da utenti particolari. Il funzionamento predefinito prevede il controllo di tutti gli utenti.

WHENEVER [NOT] SUCCESSFUL scrive una riga in una tabella di audit solo nel caso in cui un tentativo di accesso a una tabella controllata o ad una funzione di sistema abbia (o meno, NOT) successo; se questa clausola viene omessa, viene scritta una riga, a prescindere dal fatto che l'accesso abbia o meno successo.

Le informazioni di audit vengono riportate nella tabella SYS.AUD\$ e l'accesso a esse avviene attraverso le viste; vedere la voce VISTE DEL DIZIONARIO DI DATI.

### ESEMPIO

Questo esempio controlla tutti gli aggiornamenti effettuati da un utente denominato UTENTE\_IMPIEGATO:

```
audit UPDATE TABLE by utente_impiegato;
```

## AUDIT (operazione)

Le operazioni di audit rappresentano un set di funzionalità di Oracle che consentono al DBA e agli utenti di tenere traccia del funzionamento del database. Il DBA può impostare un'attività di audit predefinita. Le informazioni di audit vengono memorizzate nel dizionario dei dati e le istruzioni SQL controllano quale informazione viene memorizzata. Per esempio, è possibile ottenere una registrazione di ciascun tentativo di aggiornamento dei dati in una tabella oppure solo dei tentativi che hanno avuto successo. In alternativa, si possono registrare tutti i collegamenti effettuati a Oracle oppure soltanto i tentativi riusciti di esecuzione di operazioni riservate al DBA.

## AUTOCOMMIT

**AUTOCOMMIT** è un'impostazione di SQL\*PLUS utilizzata per effettuare il commit automatico delle modifiche apportate al database a seguito di un comando SQL del tipo `insert`, `update` o `delete`, rispettivamente, per l'inserimento, l'aggiornamento o l'eliminazione di dati nel database. Vedere SET.

## AUTORECOVERY (SQL\*PLUS)

Vedere SET.

## AVG

**VEDERE ANCHE** COMPUTE, GROUP FUNCTIONS, Capitolo 8.

### SINTASSI

`AVG ( [ DISTINCT | ALL ] espr ) [OVER ( clausola_analitica )]`

**DESCRIZIONE** AVG è la media dei valori contenuti in un gruppo di righe. DISTINCT costringe al calcolo della media solo di valori unici. Le righe NULL vengono ignorate da questa funzione, in quanto potrebbero influire sul risultato.

## AVVIO

È il processo di avvio di un'istanza, presumibilmente allo scopo di eseguire il MOUNT di un database e di aprirlo rendendolo così disponibile. Vedere il Capitolo 40.

## BACKGROUND, PROCESSO

Un processo in background è uno dei processi utilizzati da un'istanza di Oracle allo scopo di eseguire e coordinare operazioni per conto degli utenti simultaneamente collegati a un database.

## BACKUP IN LINEA

Il backup in linea è la funzione di Oracle che archivia i dati durante l'esecuzione del database. Il DBA non deve chiudere il database per archiviare i dati. Anche i dati in corso di accesso possono essere archiviati.

## BACKUP NON IN LINEA

Un backup non in linea è un backup fisico dei file del database eseguito mentre il database è chiuso.

## BEGIN

**VEDERE ANCHE** BLOCCO, STRUTTURA; DECLARE; END; EXCEPTION; TERMINATORE; Capitolo 27.

### SINTASSI

```
[<<etichetta blocco>>]
[DECLARE]
BEGIN
    ... logica del blocco ...
END [etichetta blocco];
```

**DESCRIZIONE** BEGIN è l'istruzione d'apertura della sezione esecutiva di un blocco PL/SQL. Può essere seguita da qualsiasi logica PL/SQL valida e da un gestore di eccezioni e viene chiusa dall'istruzione END. Tra BEGIN ed END è richiesta la presenza di almeno un'istruzione eseguibile. *Vedere* BLOCCO, STRUTTURA.

*etichetta blocco* è il nome attribuito a un blocco PL/SQL che inizia con la parola BEGIN e termina con la parola END. La parola END è seguita da un terminatore, solitamente un punto e virgola (*vedere* TERMINATORE per le eccezioni). *blocco* segue le convenzioni di nomenclatura normali per gli oggetti e dev'essere racchiuso tra i simboli << e >>. Questi simboli comunicano a PL/SQL che quanto vi è racchiuso rappresenta un'etichetta. BEGIN può opzionalmente essere preceduto da una sezione chiamata DECLARE (che segue l'etichetta del blocco) e può eventualmente contenere una sezione chiamata EXCEPTION.

## BETWEEN

*Vedere* OPERATORI LOGICI.

## BFILE

BFILE è un tipo di dato (*vedere* DATI, TIPI) utilizzato per dati LOB binari memorizzati all'esterno del database. Oracle non gestisce la coerenza nella lettura o l'integrità dei dati memorizzati esternamente. All'interno del database viene memorizzato il valore di un localizzatore LOB che punta al file esterno. Prima di creare una voce BFILE, è necessario creare una directory. *Vedere* CREATE DIRECTORY.

## BFILENAME

**VEDERE ANCHE** BFILE, CREATE DIRECTORY, Capitolo 32.

### SINTASSI

```
BFILENAME ( 'directory' , 'nomefile' )
```

**DESCRIZIONE** BFILENAME restituisce un localizzatore BFILE associato con il nome di file e l'oggetto di directory specificati.

## BIN\_TO\_NUM

**VEDERE ANCHE** NUMERI, FUNZIONI; Capitolo 8.

### SINTASSI

`BIN_TO_NUM ( espr [ , espr]... )`

**DESCRIZIONE** BIN\_TO\_NUM converte un vettore di bit nel suo equivalente numerico.

### ESEMPIO

```
select BIN_TO_NUM(1,0,0,1) from DUAL;
```

```
BIN_TO_NUM(1,0,0,1)
```

```
-----
```

```
9
```

## BINDING, VARIABILE

Una variabile di binding è una variabile contenuta in un'istruzione SQL che dev'essere sostituita con un valore valido o l'indirizzo di un valore in modo che l'istruzione possa essere eseguita correttamente.

## BITAND

**VEDERE ANCHE** DECODE, Capitolo 17.

### SINTASSI

`BITAND ( argomento1, argomento2 )`

**DESCRIZIONE** BITAND effettua un'operazione di calcolo AND sui bit di *argomento1* e *argomento2*, che devono entrambi risolversi in numeri non negativi, e restituisce un intero. Questa funzione viene normalmente utilizzata con la funzione DECODE.

## BITMAP, INDICE

Un indice bitmap è un tipo di indice maggiormente adatto per colonne con pochi valori distinti (e quindi, scarsa selettività). Se vi sono pochi valori distinti per una colonna e quest'ultima viene utilizzata spesso come condizione di limitazione nelle query, è opportuno considerare l'impiego di un indice bitmap per quella colonna. Vedere il Capitolo 20 per ulteriori dettagli sugli indici bitmap. Per informazioni complete riguardo alla sintassi, vedere CREATE INDEX.

## BLOB

Un BLOB è un oggetto binario di grandi dimensioni (Binary Large OBject), memorizzato all'interno del database. Vedere il Capitolo 32.

## BLOCCHI SULLE DEFINIZIONI DI DATI

Sono blocchi collocati sul dizionario dati durante le modifiche alle strutture (definizioni) degli oggetti del database (quali tabelle, indici, viste e cluster) in modo che queste modifiche si verifichino senza conseguenze negative per i dati del database. Esistono tre tipi di blocchi: i blocchi per le operazioni del dizionario, i blocchi per le definizioni del dizionario e i blocchi per le definizioni delle tabelle.

## BLOCCO

Un blocco è l'unità di base di memorizzazione (fisica e logica) per tutti i dati di Oracle. La dimensione del blocco di Oracle varia in relazione al sistema operativo e può essere diversa dalla dimensione del blocco del sistema operativo dell'host. Le dimensioni di blocco più comuni sono 4K, 8K e 16K. Per conoscere la dimensione di un blocco sul proprio sistema operativo, fare riferimento alla *Installation and User's Guide*. Le dimensioni dei blocchi del database possono differire a livello di tablespace.

## BLOCCO (LOCK)

Un blocco è una restrizione temporanea dell'accesso ai dati per gli altri utenti. La restrizione imposta a questi dati prende il nome di "lock". Le modalità di lock sono SHARE, SHARE UPDATE, EXCLUSIVE, SHARE EXCLUSIVE, ROW SHARE e ROW EXCLUSIVE. Non è possibile acquisire tutti i blocchi in tutte le modalità.

## BLOCCO A LIVELLO DI RIGA

Si tratta di un meccanismo di blocco per cui gli aggiornamenti ai dati vengono effettuati bloccando solo le righe di una tabella (e non l'intera pagina).

## BLOCCO DI INTESTAZIONE DI SEGMENTO

Il blocco di intestazione del segmento è il primo blocco nel primo extent di un segmento.

## BLOCCO DI RECORD

Il blocco (lock) di record evita che due utenti aggiormino la stessa riga di dati nello stesso istante (con le conseguenze distruttive che si possono immaginare).

## BLOCCO ESCLUSIVO

Un blocco esclusivo è un sistema di limitazione degli accessi che consente ad altri utenti di eseguire delle query sui dati pur vietando loro qualsiasi modifica. Si differenzia dal blocco SHARE in quanto non permette che altri utenti applichino qualsiasi altro tipo di limitazione sugli stessi dati; più utenti possono porre contemporaneamente blocchi SHARE sugli stessi dati.

## BLOCCO, STRUTTURA

La struttura di un blocco è la struttura delle sezioni di un blocco in PL/SQL.

**PRODOTTI** PL/SQL.

**VEDERE ANCHE** BEGIN, DECLARE, END, EXCEPTION, GOTO, Capitolo 27.

**DESCRIZIONE** I blocchi PL/SQL possono essere racchiusi in SQL\*PLUS, o in un qualunque altro linguaggio di programmazione, con il ricorso ai precompilatori di Oracle. I blocchi PL/SQL risultano strutturati nel modo seguente:

```
[<<etichetta blocco>>]
[DECLARE
.. dichiarazioni (CURSOR, VARIABLE ed EXCEPTION)...]

BEGIN
... logica blocco (codice eseguibile)...

[EXCEPTION
... logica del gestore di eccezione (per errori irreversibili)...]

END [<<etichetta blocco>>];
```

Come dimostrano le parentesi quadre, le sezioni DECLARE ed EXCEPTION sono entrambe facoltative. Un blocco può opzionalmente essere etichettato tramite un nome, che dev'essere racchiuso tra i simboli << e >>.

Le sezioni di un blocco devono essere nell'ordine specificato, anche se i blocchi possono essere annidati all'interno di altri, sia nella sezione dedicata alla logica del blocco, sia in quella del gestore delle eccezioni. I *blocchi* possono venire utilizzati per effettuare ramificazioni tramite un GOTO o come prefisso per far riferimento a una variabile contenuta in un altro blocco (*vedere* DECLARE per ulteriori dettagli in merito). Se viene fatto riferimento a una variabile nella dichiarazione di un cursore, occorre che essa sia specificata prima della dichiarazione del cursore. *Vedere* il Capitolo 27 per degli esempi riguardanti le strutture dei blocchi e i cicli.

### ESEMPIO

```
<<calc_areas>>
DECLARE
    pi      constant NUMBER(9,7) := 3.14159;
    area   NUMBER(14,2);
    cursor rad_cursor is
        select * from RADIUS_VALS;
    rad_val rad_cursor%ROWTYPE;
BEGIN
    open rad_cursor;
    loop
        fetch rad_cursor into rad_val;
        exit when rad_cursor%NOTFOUND;
        area := pi*power(rad_val.radius,2);
        insert into AREAS values (rad_val.radius, area);
    end loop;
    close rad_cursor;
END calc_areas;
```

## BREAK

**VEDERE ANCHE** CLEAR, COMPUTE, Capitoli 6 e 14.

### SINTASSI

BRE[AK] [ON *elemento\_report* [*azione* [*azione*]]] ...

Dove *elemento\_report* è:

{*colonna*|*espr*|ROW|REPORT}

e *azione* è:

[SKI[P]*n*|[SKI[P]] PAGE][NODUP[LICATES]|DUP[LICATES]]

**DESCRIZIONE** Un'interruzione ha luogo quando SQL\*PLUS rileva un cambiamento specifico, quale la fine di una pagina o la variazione nel valore di un'espressione. Essa fa in modo che SQL\*PLUS esegua alcune azioni specificate nel comando BREAK, quali SKIP, e che stampi alcuni dei risultati ottenuti tramite il comando COMPUTE, per esempio le medie dei totali di una colonna. Solo un comando BREAK per volta può essere attivo. Un nuovo comando BREAK può specificare modifiche e relative azioni associate che siano causa di un'interruzione.

Un'interruzione ON REPORT provoca un'interruzione alla fine di un report o di una query.

Una modifica nel valore di un'espressione può verificarsi in qualsiasi momento prima dell'interruzione di REPORT, e in un'unica istruzione BREAK vi possono essere diverse clausole dell'espressione ON; tuttavia il loro ordine all'interno del comando BREAK dovrebbe essere uguale a quello della clausola order by dell'istruzione select. Ciò significa che ciascuna espressione che compare nel comando BREAK dev'essere inserita anche nella clausola order by dell'istruzione select, nella stessa sequenza, altrimenti il risultato non avrà senso. In aggiunta, l'ordine dovrebbe essere dal raggruppamento più grande al più piccolo (per esempio, ON Società, ON Reparto, ON Progetto).

- ON ROW causa un'interruzione per ogni riga selezionata.
- ON PAGE provoca un'interruzione alla fine di ciascuna pagina ed è indipendente dalle interruzioni causate dall'espressione ON ROW oppure ON REPORT.
- SKIP salta un numero di *linee*, mentre PAGE o SKIP PAGE salta a una nuova pagina, prima di stampare il risultato dell'operazione COMPUTE associata all'interruzione.
- NODUPDATES sopprime la stampa, per ciascuna riga, dei valori nell'espressione o colonna all'interno di BREAK, fatta eccezione per la prima riga dopo il BREAK.
- BREAK da solo visualizza le impostazioni correnti per l'interruzione.
- CLEAR BREAKS rimuove qualsiasi BREAK esistente.

Per esempi relativi a queste opzioni, vedere il Capitolo 14.

## BTITLE (titolo a fondo pagina)

**VEDERE ANCHE** ACCEPT, DEFINE, PARAMETRI, REPFOOTER, REPHEADER, SET HEADSEP, TTITLE, Capitolo 14.

### SINTASSI

BTI[TLE] [*specstampa* [*testo*|*variabile*]... | OFF | ON]

**DESCRIZIONE** BTITLE inserisce un *testo* (anche composto da più righe) alla fine di ciascuna pagina di un report. OFF e ON rispettivamente sopprimono e ripristinano la visualizzazione del *testo* senza modificarne i contenuti. BTITLE da solo visualizza le opzioni btitle e il testo o la variabile. *testo* è un titolo di fondo pagina che si desidera aggiungere al report, mentre *variabile* rappresenta una variabile definita dall'utente o una variabile gestita dal sistema, incluse SQL.LNO, il numero della linea corrente, SQL.PNO, il numero di pagina corrente, SQL.RELEASE, il numero della versione di Oracle, SQL.SQLCODE, l'attuale codice d'errore, e SQL.USER, il nome dell'utente.

Quella che segue è una descrizione delle *opzioni*.

- COL *n* salta direttamente alla posizione *n* (dal margine sinistro) della linea corrente.
- S[KIP]*n* visualizza *n* linee vuote. Se non viene specificato alcun numero, viene stampata una sola linea vuota. Se *n* vale 0, non viene visualizzata alcuna linea vuota e la posizione di stampa corrente diventa la prima della linea corrente (all'estremità sinistra della pagina).
- TAB *n* salta di *n* tabulazioni verso destra (verso sinistra se *n* è negativo).
- BOLD stampa i dati dell'output in grassetto.
- LE[FT], CE[NTER] e RI[GHT] rispettivamente, allineano a sinistra, centrano e allineano a destra i dati sulla riga corrente. Eventuali stringhe di testo e variabili che seguissero questi comandi verrebbero giustificate anch'esse, fino alla fine del comando. CENTER e RIGHT utilizzano il valore impostato da SET LINESIZE per determinare dove disporre il testo e le variabili.
- La stringa FORMAT specifica il modello che controlla il formato del testo o delle variabili che seguono ed è sottoposto alla stessa sintassi dell'opzione FORMAT del comando COLUMN, come FORMAT A12 oppure FORMAT \$999,999.99. Ogni volta che compare FORMAT, esso scavalca il precedente formato attivo fino a quel momento. Se non viene specificato alcun modello per FORMAT, viene utilizzato quello impostato da SET NUMFORMAT. Se NUMFORMAT non è stato definito, viene utilizzato il modello predefinito per SQL\*PLUS.

I valori che si riferiscono a date vengono stampati in conformità con il formato predefinito, a meno che non sia stata caricata una variabile con una data riformattata tramite TO\_CHAR.

In un unico btitle può essere specificato un numero qualsiasi di opzioni, testi e variabili. Ognuno viene stampato nell'ordine specificato e viene formattato in base alle clausole che lo precedono.

## B-TREE

B-TREE rappresenta una struttura di indicizzazione utilizzata da Oracle per creare e memorizzare gli indici.

## BUFFER

In termini generali, un *buffer* è una specie di blocco degli appunti, solitamente contenuto nella memoria del computer, in cui i comandi vengono posizionati temporaneamente per la modifica e l'esecuzione.

In SQL\*PLUS, un buffer rappresenta un'area di memoria per la modifica di comandi SQL. Vedere EDIT e SET.

## BUFFER (DATABASE)

I buffer del database sono spazi temporanei di memorizzazione di blocchi del database utilizzati in quel momento dagli utenti.

## **BUFFER (REDO LOG)**

I buffer di redo log sono spazi temporanei di memorizzazione di voci redo log create dalle transazioni che avvengono all'interno del database.

C, LINGUAGGIO

Il C è un linguaggio di programmazione famoso per la sua portabilità verso molti tipi differenti di computer. Oracle stesso è scritto principalmente in C.

## CACHE

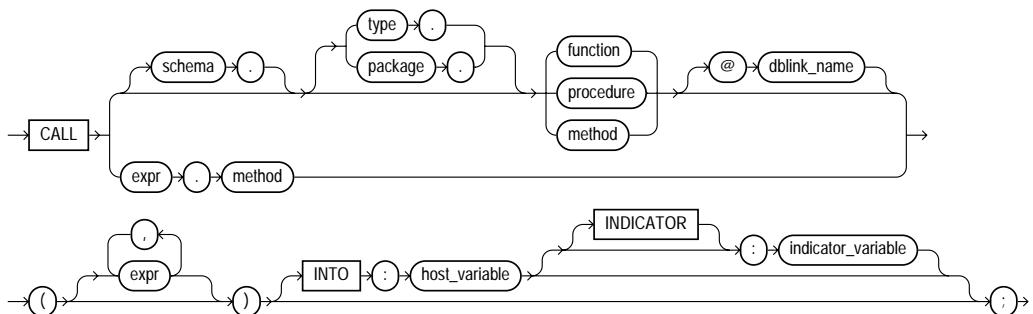
Le cache sono spazi temporanei di memorizzazione sia per i dati del database utilizzati (tramite operazioni di accesso o modifica) in quel momento dagli utenti, sia per i dati richiesti da Oracle per il supporto agli utenti.

## CALL

**VEDERE ANCHE** EXECUTE, Capitolo 36.

SINTASSI

call



**DESCRIZIONE** È possibile utilizzare CALL per eseguire una stored procedure o una funzione da SQL. Occorre però disporre del privilegio EXECUTE sulla procedura o funzione, o sul package nel quale la procedura o funzione esiste.

CAMPO

In una tabella, un campo equivale all'informazione contenuta all'intersezione di una riga e di una colonna. Generalmente, campo è sinonimo di colonna, sebbene possa riferirsi spesso ai valori in essa contenuti.

## CARATTERI, FUNZIONI

**VEDERE ANCHE** CONVERSIONE, FUNZIONI; NUMERI, FUNZIONI; ALTRE FUNZIONI; Capitolo 7.

**DESCRIZIONE** Riportiamo un elenco alfabetico di tutte le funzioni per caratteri di SQL di Oracle. Ciascuna viene riportata altrove in questa guida, assieme alla sintassi e ad esempi d'uso.

### Nomi e modalità di utilizzo delle funzioni

`stringa || stringa`

|| concatena due stringhe. Il simbolo | viene chiamato barra verticale spezzata, anche se in alcuni computer può comparire come una barra intera.

`ASCII(stringa)`

**ASCII** fornisce il valore ASCII del primo carattere di una stringa.

`CHR(intero)`

**CHR** fornisce il carattere il cui valore ASCII equivale a un dato intero positivo.

`CONCAT(stringa1,stringa2)`

**CONCAT** concatena due stringhe. Equivale a ||.

`INITCAP(stringa)`

Sta per INITial CAPital (iniziale maiuscola). Rende maiuscola la prima lettera di una parola.

`INSTR(stringa, set [, inizio [, occorrenza ] ])`

**INSTR** trova la posizione dell'inizio di un insieme di caratteri IN una STRinga.

`LENGTH(stringa)`

Restituisce la lunghezza di una stringa.

`LOWER(stringa)`

**LOWER** converte ciascuna lettera di una stringa in caratteri minuscoli.

`LPAD(stringa, lunghezza [, 'car'])`

**LPAD** sta per Left PAD (riempimento a sinistra). Rende una stringa di una lunghezza determinata, aggiungendo a sinistra la sequenza di caratteri specificata da *car*.

`LTRIM(stringa [, 'car'])`

**LTRIM** sta per Left TRIM. Taglia dalla parte di sinistra di una stringa tutte le occorrenze della sequenza di caratteri specificata da *car*.

`NLS_INITCAP(stringa[, 'NLS_SORT=ordine'])`

NLS\_INITCAP sta per National Language Support INInitial CAPital. Questa versione di INITCAP utilizza l'ordine della sequenza di confronto per effettuare la conversione tra maiuscole e minuscole.

`NLS_LOWER(stringa, 'NLS_SORT=ordine'])`

NLS\_LOWER sta per National Language Support LOWER case. Questa versione di LOWER utilizza l'ordine della sequenza di confronto per effettuare la conversione tra maiuscole e minuscole.

`NLS_UPPER(stringa[, 'NLS_SORT=ordine'])`

NLS\_UPPER sta per National Language Support UPPER case. Questa versione di UPPER utilizza l'ordine della sequenza di confronto per effettuare la conversione tra maiuscole e minuscole.

`NLSSORT(stringa[, 'NLS_SORT=ordine'])`

Oracle utilizza il National Language Support SORT. Questa funzione fornisce il valore della sequenza di confronto della stringa data nella sequenza di confronto *ordine* o, se quest'ultima viene omessa, l'opzione National Language Support scelta per il sito.

`REPLACE(stringa, if [,then])`

REPLACE restituisce la *stringa* dopo che ciascuna occorrenza di *if* è stata sostituita con *then* (zero o più caratteri). Se non viene specificata alcuna stringa *then*, vengono rimosse tutte le occorrenze di *if*. Vedere TRANSLATE.

`RPAD(stringa, lunghezza [, 'car'])`

RPAD sta per Right PAD. Rende una stringa di una lunghezza specifica aggiungendo a destra un determinato insieme di caratteri.

`RTRIM(stringa [, 'car'])`

RTRIM sta per Right TRIM. Taglia tutte le occorrenze della sequenza di caratteri specificata da *car* dalla parte destra di una stringa.

`SOUNDEX(stringa)`

SOUNDEX converte una stringa in un valore di codice. I nomi con fonema simile tendono ad avere lo stesso valore di codice. È possibile utilizzare SOUNDEX per confrontare i nomi che possono avere piccole differenze di pronuncia, ma sono di fatto lo stesso nome (vale per la lingua inglese).

`SUBSTR(stringa, inizio [,conta])`

SUBSTR ritaglia una parte di una stringa a partire dalla posizione *inizio* e contando un numero *conta* di caratteri a partire da *inizio*.

`TRANSLATE(stringa,if,then)`

Questa funzione traduce una stringa, carattere per carattere, in base a una corrispondenza di posizioni di caratteri nella stringa *if* con caratteri nella stringa *then*. Vedere REPLACE.

**TRIM**

```
( [ { { LEADING | TRAILING | BOTH } [caratteri_da_tagliare]
      | caratteri_da_tagliare
    }   FROM   ]   sorgente_da_tagliare )
```

**TRIM** taglia tutte le occorrenze della sequenza di caratteri specificata da *car* dalla parte destra, da quella sinistra o da entrambe le parti di una stringa.

**UPPER(*stringa*)**

**UPPER** converte ciascuna lettera di una stringa in maiuscolo.

**USERENV(*opzione*)**

**USERENV** restituisce informazioni relative all'ambiente dell'utente, solitamente per una coda di audit. Le opzioni sono **ENTRYID**, **SESSIONID** e **TERMINAL**. **USERENV** è ancora supportata ma è stata sostituita dallo spazio di nomi UserEnv di **SYS\_CONTEXT**.

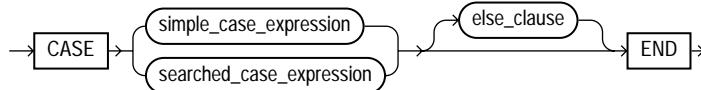
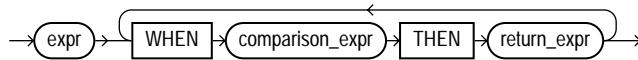
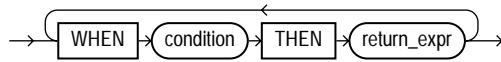
**VSIZE(*stringa*)**

**VSIZE** fornisce la dimensione di memorizzazione di *stringa* in Oracle.

## CASE

**VEDERE ANCHE** ALTRE FUNZIONI, DECODE, TRANSLATE, Capitolo 17.

### SINTASSI

**case\_expression****simple\_case\_expression****searched\_case\_expression****else\_clause**

**DESCRIZIONE** CASE supporta la logica if-then-else nelle istruzioni SQL. È possibile utilizzare le clausole WHEN/THEN per indicare al database cosa fare per ciascuna condizione. È possibile impiegare una clausola ELSE per gestire le eccezioni alle proprie clausole WHEN.

## ESEMPIO

```
select distinct
CASE NomeCategoria
when 'NARRADULTI' then 'Narrativa adulti'
when 'SAGGIADULTI' then 'Saggi adulti'
when 'CONSADULTI' then 'Consultazione adulti'
when 'NARRARAGAZZI' then 'Narrativa ragazzi'
when 'SAGGIRAGAZZI' then 'Saggi ragazzi'
when 'ILLUSRAGAZZI' then 'Illustrati ragazzi'
else NomeCategoria
end
from BIBLIOTECA;
```

## CAST

**VEDERE ANCHE** Vedere TABELLA ANNIDATA, Capitolo 31.

### SINTASSI

CAST ( { *expr* | ( *subquery* ) | MULTiset ( *subquery* ) } AS *nome\_tipo* )

**DESCRIZIONE** CAST può essere utilizzato per convertire un tipo di dato in un altro. CAST viene utilizzato più comunemente quando si lavora con dei collettori come le tabelle annidate.

## CATSEARCH

CATSEARCH viene impiegato per le ricerche di testo in indici testuali CTXCAT creati in Oracle Text. Consultare il Capitolo 24 per esempi di creazione di indici e di ricerche di testo. Tra gli operatori di ricerca del testo utilizzabili si segnalano i seguenti.

OPERATORE	DESCRIZIONE
	Restituisce un record se almeno uno dei termini di ricerca ha un punteggio superiore alla soglia.
AND	Restituisce un record se entrambi i termini di ricerca hanno un punteggio superiore alla soglia. Questa è l'azione predefinita.
-	Restituisce le righe che contengono il termine che precede l'operatore “-” e che non contengono il termine che lo segue.
NOT	Identico a -.
()	Specifica l'ordine di valutazione dei criteri di ricerca.
“ ”	Racchiude delle frasi.

## CEIL

**VEDERE ANCHE** FLOOR; NUMERI, FUNZIONI.

### SINTASSI

CEIL(*valore*)

**DESCRIZIONE** CEIL è il più piccolo intero maggiore o uguale a *valore*.

### ESEMPIO

```
CEIL(1.3) = 2
CEIL(-2.3) = -2
```

## CATENA, RIGA IN

Una riga in catena è una riga che viene memorizzata in più di un blocco di database e che pertanto è composta da più parti. Le righe lunghe (o LOB in linea), la cui lunghezza è maggiore della dimensione di un blocco possiedono sempre vari pezzi di riga. Il comando ANALYZE può identificare le righe in catena e fornire informazioni statistiche sul loro numero. Vedere ANALYZE.

## CHANGE

**VEDERE ANCHE** APPEND, DEL, EDIT, LIST, Capitolo 6.

### SINTASSI

```
C[CHANGE] /vecchio testo/nuovo testo/
```

**DESCRIZIONE** CHANGE è una funzionalità dell'editor a linea di comando di SQL\*PLUS. Consente di modificare un *vecchio testo* in un *nuovo testo* all'interno della linea corrente del buffer in uso (la linea segnata con un \* nell'elenco).

CHANGE ignora la distinzione tra maiuscole e minuscole nella ricerca del testo vecchio. Tre punti rappresentano un carattere jolly. Se *vecchio testo* viene fatto precedere da '...', qualsiasi cosa fino alla prima occorrenza, inclusa, del vecchio testo, viene sostituita con il nuovo testo. Se a *vecchio testo* vengono accodati i '...', qualsiasi cosa successiva alla prima occorrenza, inclusa, del vecchio testo viene sostituita dal nuovo testo. Se *vecchio testo* racchiude i ..., qualsiasi cosa a partire dalla parte del vecchio testo precedente ai tre punti, fino alla parte del vecchio testo dopo di essi, viene sostituita con il nuovo testo.

Lo spazio tra CHANGE e il primo simbolo / può essere omesso; il delimitatore finale non è necessario se non è richiesto l'inserimento di alcuno spazio in coda. Può essere utilizzato un delimitatore diverso da /. Si suppone che qualunque carattere successivo alla parola CHANGE (diverso dallo spazio) sia il delimitatore.

### ESEMPI

Se questa è la riga corrente del buffer in uso:

```
where NomeCategoria='NARRADULTI'
```

questa riga:

```
C /NARRADULTI/SAGGIADULTI
```

produce l'output seguente:

```
where NomeCategoria='SAGGIADULTI'
```

## CHAR, TIPO DI DATI

Vedere DATI, TIPI.

## CHARTOROWID

Vedere anche CONVERSIONE, FUNZIONI; ROWIDTOCHAR.

### SINTASSI

CHARTOROWID(stringa)

**DESCRIZIONE** CHARTOROWID sta per CHARacter TO ROW IDentifier (da carattere a identificatore di riga). Essa modifica una stringa di caratteri in modo che funzioni come un identificatore di riga interno di Oracle o ROWID.

## CHECKPOINT

Un checkpoint è un punto determinato nel quale i blocchi modificati vengono scritti nel database dalla SGA.

## CHIAMATA RICORSIVA

Una chiamata ricorsiva è una chiamata annidata dell'RDBMS; per esempio, le informazioni di audit sono registrate nelle tabelle di sistema tramite chiamate ricorsive. Durante una normale operazione di database, magari un aggiornamento, viene eseguita un'altra operazione per scrivere informazioni di log, di audit e altre informazioni vitali *relative* alle normali operazioni di database in corso.

## CHIAVE

Una chiave è una colonna (o più colonne) utilizzata per identificare delle righe; non equivale a un indice, anche se spesso vengono utilizzati insieme. Vedere CHIAVE ESTERNA, CHIAVE PRIMARIA e CHIAVE UNICA.

## CHIAVE COMPOSTA

Si tratta di una chiave primaria o esterna composta da due o più colonne.

## CHIAVE ESTERNA

Una chiave esterna è una colonna, o un insieme di colonne, i cui valori si basano su quelli delle chiavi primarie o candidate di un'altra tabella.

## CHIAVE PRIMARIA

La chiave primaria è quella colonna (ma possono essere anche più d'una) che identifica in modo univoco ogni riga di una tabella.

## CHIAVE UNICA

Una chiave unica è una colonna (o più colonne) che deve contenere valori unici per ciascuna riga della tabella. Vedere CHIAVE, CHIAVE PRIMARIA e VINCOLI DI INTEGRITÀ.

## CHIUSURA (SHUTDOWN)

Questo termine indica la disconnessione di un'istanza dal database e la sua successiva conclusione. Vedere (e fare un confronto con) START. In qualità di comando di tipo Server Manager, le opzioni di SHUTDOWN sono NORMAL (chiusura normale), IMMEDIATE (chiusura immediata) e ABORT (chiusura interrotta). Vedere il Capitolo 40.

## CHR

**VEDERE ANCHE** ASCII; CARATTERI, FUNZIONI.

### SINTASSI

CHR(*intero*)

**DESCRIZIONE** CHR restituisce il carattere a cui corrisponde il valore ASCII di *intero* (*intero* rappresenta un numero intero tra 0 e 255, poiché il valore ASCII di un carattere è un intero tra 0 e 255). I valori tra lo 0 e il 127 possiedono standard ben definiti. I valori superiori al 127 (chiamati set esteso di caratteri ASCII) tendono a differire da nazione a nazione, in relazione all'applicazione e al costruttore di computer. La lettera A, per esempio, è uguale al numero ASCII 65, B è 66, C è 67 e così via. La virgola decimale è 46. Il segno meno è 45. Il numero 0 è 48, 1 è 49, 2 è 50 e così via.

### ESEMPIO

```
select CHR(77), CHR(46), CHR(56) from DUAL;
```

```
C C C  
- - -  
M . 8
```

## CICLI

È possibile utilizzare dei cicli per elaborare più record all'interno di un unico blocco PL/SQL. Questo linguaggio supporta tre tipi di cicli:

Cicli semplici	Tutti i cicli che continuano a ripetersi fino al raggiungimento di un'istruzione exit o exit when.
Cicli FOR	I cicli che si ripetono un determinato numero di volte.
Cicli WHILE	Cicli che si ripetono fino al raggiungimento di una condizione.

I cicli possono elaborare record multipli a partire da un cursore. Il ciclo a cursore più comune è il ciclo FOR. Per dettagli riguardanti l'uso di cicli sia per la logica di elaborazione semplice sia per quella basata sui cursori, si *consulti* CICLI A CURSORI e il Capitolo 27.

## CICLI A CURSORE

Nei cicli FOR le iterazioni vengono eseguite per un numero specifico di volte. In un ciclo FOR a cursore, i risultati di una query vengono utilizzati per determinare in modo dinamico quante volte il ciclo dovrà essere eseguito. In un ciclo FOR a cursore, l'apertura, l'operazione di fetch e la chiusura dei cursori viene effettuata in modo implicito: non è necessario codificare in modo esplicito queste azioni.

Nel listato seguente è possibile vedere un ciclo FOR a cursore che esegue una query sulla tabella VALORI\_RAGGIO e inserisce dei record nella tabella AREE.

```
declare
    pi      constant NUMBER(9,7) := 3.1415927;
    area   NUMBER(14,2);
    cursore rag_cursore is
        select * from VALORI_RAGGIO;
    val_rag_ rag_cursore%ROWTYPE;
begin
    for val_rag in rag_cursore
    loop
        area := pi*power(val_rag.raggio,2);
        insert into AREE values (val_rag.raggio, area);
    end loop;
end;
```

In un ciclo FOR a cursore non esistono comandi open o fetch. Il comando:

```
for val_rag in rag_cursore
```

apre implicitamente il cursore “*rag\_cursore*” e trasmette un valore nella variabile “*val\_rag*”. Si noti che la variabile *val\_rag* non viene dichiarata esplicitamente quando si usa un ciclo FOR a cursore. Quando nel cursore non ci sono più record, il ciclo viene terminato e il cursore chiuso. In un ciclo FOR a cursore non vi è necessità di un comando close. Vedere il Capitolo 27 per ulteriori informazioni sulla gestione di un cursore all'interno di PL/SQL.

## CLAUSOLA

Una clausola è una delle parti principali di un'istruzione SQL che inizia con una parola del tipo select, insert, update, delete, from, where, order by, group by o having.

## CLEAR

**VEDERE ANCHE** BREAK, COLUMN, COMPUTE.

### SINTASSI

```
CL[EAR] opzione
```

**DESCRIZIONE** CLEAR azzera tutte le opzioni.

BRE[AKS] elimina le interruzioni impostate tramite il comando BREAK.

BUFF[ER] azzera il buffer corrente.

COL[UMNS] azzera le opzioni impostate tramite il comando COLUMN.

COMP[UTES] azzerà le opzioni impostate tramite il comando COMPUTE.

SCR[EEN] ripulisce lo schermo.

SQL azzerà il buffer SQL.

TIMI[NG] elimina tutte le aree di temporizzazione create dal comando TIMING.

### ESEMPI

Per azzerare i calcoli, eseguire il seguente comando:

```
clear computes
```

Per azzerare le definizioni delle colonne, il comando è:

```
clear columns
```

## CLIENT

Client è un termine generale che indica un utente, un'applicazione software o un computer che richiede i servizi, i dati o il processo di un'altra applicazione o di un altro computer.

## CLOB

CLOB è un tipo di dato che supporta oggetti carattere di grandi dimensioni (character large object). Vedere il Capitolo 32.

## CLOSE

**VEDERE ANCHE** DECLARE, FETCH, FOR, OPEN, Capitolo 27.

### SINTASSI

```
CLOSE cursoro;
```

**DESCRIZIONE** CLOSE chiude il cursore specificato e ne rilascia le risorse in modo che Oracle le possa riutilizzare altrove; *cursoro* dev'essere il nome di un cursore attualmente aperto.

Anche se un cursore è stato chiuso, la sua definizione non è perduta ed è possibile eseguire nuovamente OPEN cursore, a patto che precedentemente sia stato dichiarato in modo esplicito. Un ciclo FOR apre in modo implicito un cursore dichiarato. Vedere CICLI A CURSORE.

## CLUSTER

Un cluster rappresenta un mezzo per memorizzare assieme i dati provenienti da più tabelle, quando i dati delle tabelle contengono informazioni in comune ed è probabile che si acceda ad esse simultaneamente. È anche possibile avere un cluster con una singola tabella. Vedere CREA-TÉ CLUSTER e il Capitolo 20.

## CLUSTER, CHIAVE

La chiave di cluster è la colonna o le colonne che le tabelle associate a cluster hanno in comune e che viene scelta come chiave di memorizzazione/accesso. Per esempio, le due tabelle BIBLIO-

TECA e AUTORE\_BIBLIOTECA possono essere associate a cluster sulla colonna Titolo. La chiave di un cluster è uguale alla colonna di un cluster.

## **CLUSTER HASH**

Un cluster hash è un cluster memorizzato da una chiave di hash invece che da una chiave di indice. Una chiave di hash è un valore calcolato a partire dai valori delle chiavi che rappresentano la locazione su disco. Una chiave di indice richiede che Oracle cerchi la locazione nell'indice, mentre una chiave di hash consente a Oracle di calcolare la locazione.

## **CLUSTER, INDICE**

L'indice di un cluster viene creato manualmente dopo la creazione del cluster e prima che qualsiasi istruzione DML (select, insert, update o delete) possa operare sul cluster. Questo indice viene creato sulle colonne chiave del cluster tramite l'istruzione SQL CREATE INDEX. In Oracle è possibile definire un cluster hash per indicizzare la chiave primaria. *Vedere CLUSTER HASH.*

## **CMDSEP (SQL\*PLUS)**

*Vedere SET.*

## **COALESCE**

**VEDERE ANCHE** DECODE, Capitolo 17.

### **SINTASSI**

COALESCE(*valore1, valore2, ...*)

**DESCRIZIONE** COALESCE restituirà il primo valore non NULL incontrato nell'elenco dei valori forniti.

## **COALESCE (spazio)**

Questa opzione consente di unire extent liberi adiacenti in un unico extent. Per esempio, se due extent di 100 blocchi sono vicini l'uno all'altro all'interno di una tablespace, possono essere riuniti in un singolo extent di 200 blocchi. Il processo in background SMON consente di unire spazi liberi all'interno di tablespace il cui valore pctincrease predefinito è diverso da zero. È possibile riunire manualmente dello spazio libero all'interno di una tablespace tramite l'opzione coalesce del comando alter tablespace. *Vedere ALTER TABLESPACE.*

## **CODA DI AUDIT**

Una coda di audit è un record fisico delle righe che vengono scritte quando la funzionalità di audit è abilitata. Se l'audit è stato abilitato (tramite il parametro di inizializzazione AUDIT\_TRAIL) e dei comandi di audit sono stati impartiti dagli utenti o dal DBA, la tabella della coda di audit interna AUD\$, di proprietà di SYS, contiene tutte le righe della coda di audit.

## COERENZA DI LETTURA

La coerenza di lettura è uno stato che garantisce che tutti i dati di un'istruzione o di una transazione rimangano coerenti per tutta la durata dell'istruzione/transazione. *Vedere SET TRANSACTION.*

## COLONNA

Una colonna rappresenta una suddivisione di una tabella identificata da un nome e da un tipo di dato. Per esempio, in una tabella contenente i dati dei dipendenti di una ditta, tutte le età dei dipendenti costituiscono una colonna. *Vedere RIGA.*

## COLONNA DI JOIN

Una colonna di join è una colonna utilizzata nell'unione di una tabella con un'altra. Una colonna di join può essere identificata usandola in una clausola where, in una clausola using oppure in una clausola on che specifichi la relazione tra le due tabelle.

## COLONNA, VINCOLO

Un vincolo su una colonna è un vincolo integrità assegnato a una specifica colonna di una tabella. *Vedere VINCOLO DI INTEGRITÀ.*

## COLSEP (SQL\*PLUS)

*Vedere SET.*

## COLUMN

**VEDERE ANCHE** ALIAS, Capitolo 6 e Capitolo 14.

### SINTASSI

```
COL[UMN]      {colonna | espressione}
  [ ALI[AS] alias ]
  [ CLE[AR] | DEF[AULT] ]
  [ ENTMAP {ON|OFF} ]
  [ FOLD_A[FTER] ]
  [ FOLD_B[EFORE] ]
  [ FOR[MAT] formato ]
  [ HEA[DING] testo
    [ JUS[TIFY] {L[EFT]|C[ENTER]|C[ENTRE]|R[IGHT]} ] ]
  [ LIKE {espressione | etichetta} ]
  [ NEWL[INE] ]
  [ NEW_V[ALUE] variabile ]
  [ NOPRI[NT]|PRI[NT] ]
  [ NUL[L] testo ]
  [ ON | OFF ]
  [ OLD_V[ALUE] variabile ]
  [ WRA[PPED]|WOR[D_WRAPPED]|TRU[NCATED] ]...
```

**DESCRIZIONE** COLUMN controlla le colonne e la formattazione dell'intestazione della colonna. Le opzioni sono tutte cumulative e possono essere inserite sia simultaneamente su una singola linea, sia su linee separate in qualsiasi momento; l'unico requisito è che la parola COLUMN e la colonna o l'espressione appaiano su una linea separata. Se una delle opzioni è ripetuta, risulta attiva quella specificata più di recente. COLUMN da solo visualizza tutte le definizioni correnti per ciascuna colonna. COLUMN con solo la specifica di una colonna o di un'espressione visualizza la definizione corrente di quella colonna.

*colonna* o *espressione* fa riferimento a una colonna o ad un'espressione utilizzata nell'istruzione select. Se viene utilizzata un'espressione, quest'ultima dev'essere inserita esattamente nello stesso modo in cui compare nell'istruzione select. Se l'espressione in select è Importo\*Aliquota e nel comando COLUMN si inserisce Aliquota \* Importo, il comando non funzionerà. Se nell'istruzione select viene assegnato un alias alla colonna o espressione, è necessario utilizzare quell'alias.

Se da tabelle differenti si selezionano colonne aventi lo stesso nome (in select sequenziali), un comando COLUMN per quel nome viene applicato a ciascuna colonna con quel nome. Per evitare ciò, all'interno dell'istruzione select è sufficiente assegnare alle colonne alias differenti (non con la clausola alias del comando COLUMN) e scrivere un comando COLUMN per ciascun alias della colonna.

ALIAS assegna alla colonna un nuovo nome, che può essere utilizzato come riferimento alla colonna nei comandi BREAK e COLUMN.

CLEAR elimina la definizione della colonna.

DEFAULT lascia la colonna definita e nello stato ON, ma elimina qualsiasi altra opzione.

ENTMAP consente l'attivazione o disattivazione del mapping delle entità per le colonne selezionate nell'output HTML.

FOLD\_A[FTER] e FOLD\_B[EFORE] istruisce Oracle a stampare su più righe una singola riga di output. È possibile scegliere di interrompere la riga prima o dopo la colonna.

FORMAT specifica il formato di visualizzazione della colonna. Il formato dev'essere un valore letterale quale A25 o 990.99. Senza un formato specifico, l'ampiezza della colonna coincide con la lunghezza secondo quanto definito nella tabella.

L'ampiezza di una colonna di tipo LONG ha un valore predefinito pari a SET LONG. È possibile impostare l'ampiezza sia dei campi CHAR che LONG normali tramite un formato del tipo FORMAT An, dove *n* rappresenta un intero con la nuova dimensione della colonna.

L'ampiezza di una colonna numerica ha come valore predefinito SET NUMWIDTH, ma viene modificata dall'ampiezza in una clausola format, quale FORMAT 999,999.99. Queste opzioni funzionano sia con il comando set numformat che con column format:

FORMATO	RISULTATO
9999990	Il conteggio dei nove e zeri determina le cifre massime che possono essere visualizzate.
999,999,999.99	Le virgolette e i decimali vengono specificati secondo il modello mostrato.
999990	Visualizza zero se il valore è zero.
099999	Visualizza i numeri con degli zeri iniziali.
\$99999	Un segno di dollaro inserito all'inizio di ciascun numero.
B99999	Non viene visualizzato niente se il valore è zero.
99999MI	Se il numero è negativo, un segno meno segue il numero. Per default il segno negativo è sulla sinistra.

(segue)

(continua)

FORMATO	RISULTATO
S9999	Restituisce "+" per i valori positivi, "-" per quelli negativi.
99999PR	I numeri negativi vengono visualizzati compresi tra < e >.
99D99	Visualizza il decimale nella posizione indicata.
9G999	Visualizza il separatore di gruppo nella posizione indicata.
C9999	Visualizza il simbolo valutario ISO nella posizione indicata.
L999	Visualizza il simbolo di valuta locale.
,	Visualizza una virgola.
.	Visualizza un punto.
9.999EEEE	La rappresentazione avviene in notazione scientifica (devono esserci esattamente 4 E).
999V99	Moltiplica il numero per $10^n$ , dove $n$ rappresenta il numero di cifre a destra di V. 999V99 traduce 1234 in 123400.
RN	Visualizza il numerale romano per i numeri compresi tra 1 e 3999.
DATE	Visualizza il valore come data nel formato MM/DD/YY per le colonne NUMBER utilizzate per memorizzare date del calendario Giuliano.

**HEADING** riassegna un'etichetta all'intestazione di una colonna. Il valore predefinito è rappresentato dal nome della colonna o dall'espressione. Se il testo possiede spazi o caratteri di punteggiatura, dev'essere racchiuso tra apici singoli. Il carattere **HEADSEP** (in genere '|') all'interno di un testo fa in modo che SQL\*PLUS inizi una nuova linea. Il comando **COLUMN** ricorda il carattere **HEADSEP** corrente quando la colonna viene definita e continua a utilizzarlo per la colonna in questione fino a che quest'ultima non viene ridefinita, anche se il carattere **HEADSEP** risulta modificato.

**JUSTIFY** allinea l'intestazione al di sopra della colonna. Per default è pari a **RIGHT** (allineamento a destra) per le colonne numeriche e **LEFT** per qualunque altro elemento.

**LIKE** riproduce le definizioni di una colonna definita in precedenza per la colonna attuale, laddove l'espressione o l'etichetta sono state utilizzate nella definizione dell'altra colonna. Vengono copiate solo le funzionalità dell'altra colonna che non sono state definite in modo esplicito.

**NEWLINE** inizia una nuova riga prima di stampare il valore della colonna.

**NEW\_VALUE** definisce una variabile che contenga il valore della colonna da utilizzare nel comando **ttitle**. Vedere il Capitolo 14 per informazioni relative all'utilizzo di questo comando.

**NOPRINT** e **PRINT** abilita o disabilita la visualizzazione della colonna.

**NULL** imposta il testo da visualizzare se la colonna possiede un valore **NULL**. L'impostazione predefinita per questa opzione è una stringa di spazi di larghezza pari a quella della colonna.

**OFF** o **ON** abilita o disabilita tutte queste opzioni per una colonna, senza influenzarne i contenuti.

**OLD\_VALUE** definisce una variabile che contenga il valore della colonna da utilizzare nel comando **btitle**. Vedere il Capitolo 13 per informazioni relative all'utilizzo di questo comando.

**WRAPPED**, **WORD\_WRAPPED** e **TRUNCATED** controllano il modo in cui SQL\*PLUS visualizza un'intestazione o il valore di una stringa troppo ampio per essere contenuto nella

colonna. WRAP prosegue la scrittura del valore alla riga successiva. WORD\_WRAP esegue un'operazione simile, ma fa cadere l'interruzione sulle parole. TRUNCATED tronca il valore all'ampiezza della definizione della colonna.

## COMANDO

*Vedere ISTRUZIONE.*

## COMANDO, LINEA DI

La linea di comando è la riga sullo schermo del computer dove vengono inseriti i comandi.

## COMMENT

**VEDERE ANCHE** VISTE DEL DIZIONARIO DI DATI, Capitolo 37.

### SINTASSI

```
COMMENT ON
{ TABLE [schema .] { tabella | vista | vista materializzata }
| COLUMN [schema .] {tabella . | vista . | vista materializzata . } colonna
| OPERATOR [schema .] operatore
| INDEXTYPE [schema .] tipoindice
}
IS 'testo';
```

**DESCRIZIONE** COMMENT inserisce nel dizionario di dati il testo di commento relativo a un oggetto o ad una colonna.

Per eliminare un commento dal database è sufficiente impostarlo al valore di una stringa vuota (set testo to "").

## COMMIT

L'espressione "effettuare il commit" significa apportare modifiche ai dati (tramite `insert`, `update` e `delete`) in modo permanente. Prima che le modifiche vengano memorizzate, esistono sia i vecchi che i nuovi dati, perciò è possibile rendere effettive le modifiche oppure ripristinare i dati originari (`rollback` dei dati). Quando un utente inserisce il comando COMMIT di Oracle SQL, tutte le modifiche provenienti da quella transazione vengono rese permanenti.

## COMMIT (forma 1, interno SQL)

**VEDERE ANCHE** ROLLBACK, SAVEPOINT, SET TRANSACTION, guida di programmazione del precompilatore.

### SINTASSI

```
EXEC SQL [AT { database| :variabile host}] COMMIT [WORK]
[ [COMMENT 'testo' ] [RELEASE]
| FORCE 'testo' [. intero ]]
```

**DESCRIZIONE** Si utilizza COMMIT per suddividere il lavoro in varie fasi all'interno di un programma. Senza l'utilizzo esplicito di COMMIT, il lavoro di un intero programma viene considerato come un'unica transazione e non viene registrato fino a che il programma ha termine. Qualsiasi lock viene mantenuto fino alla fine, impedendo agli altri utenti l'accesso ai dati bloccati. COMMIT andrebbe utilizzato ogni volta che è logicamente praticabile.

WORK è facoltativo e non ha effetti sull'utilizzo; viene fornito per la compatibilità ANSI. AT fa riferimento a un database remoto a cui si è avuto accesso tramite il comando DECLARE DATABASE. RELEASE disconnette l'utente dal database, sia remoto sia locale. FORCE effettua il commit manuale di una transazione distribuita sconosciuta.

## COMMIT (forma 2, istruzione PL/SQL)

**VEDERE ANCHE** ROLLBACK, SAVEPOINT.

### SINTASSI

```
COMMIT [WORK] [ COMMENT 'testo' | FORCE 'testo' [, intero] ];
```

**DESCRIZIONE** COMMIT rende effettive le modifiche apportate al database dal momento in cui è stato eseguito l'ultimo COMMIT, sia隐式的还是显式. WORK è facoltativo e non ha effetti sull'utilizzo.

COMMENT associa alla transazione un testo di commento che può essere visualizzato tramite la vista del dizionario di dati DBA\_2PC\_PENDING nel caso in cui una transazione distribuita non possa essere completata. FORCE effettua il commit manuale di una transazione distribuita sconosciuta.

## COMMIT A DUE FASI

Oracle gestisce le transazioni distribuite mediante una caratteristica speciale chiamata *commit a due fasi*. Questo meccanismo garantisce che le transazioni risultino valide su tutti i siti, dal momento in cui viene eseguito il commit o il rollback. Tutti i siti salvano o annullano contemporaneamente, indipendentemente dall'errore verificatosi sulla rete o sui computer interconnessi tramite rete. Il commit a due fasi è un meccanismo attivo per default.

## COMPOSE

**VEDERE ANCHE** CONVERSIONE, FUNZIONI; Capitolo 10.

### SINTASSI

```
COMPOSE(stringa)
```

**DESCRIZIONE** COMPOSE prende come argomento una stringa in qualunque tipo di dato e restituisce una stringa UNICODE nella sua forma completamente normalizzata utilizzando il medesimo set di caratteri della stringa di input.

### ESEMPIO

Per visualizzare una “o” con la dieresi:

```
select COMPOSE( 'o' || UNISTR('\0308') ) from DUAL;
```

C  
-  
ö

## COMPUTE

**VEDERE ANCHE** BREAK, FUNZIONI DI GRUPPO.

### SINTASSI

```
COMP[UTE][AVG|COU[NT]|MAX[IMUM]|MIN[IMUM]|NUM[BER]|STD|SUM|VAR[IANCE]]...
  [funzione LABEL nome_etichetta
   OF {espressione | colonna | alias} ...
   ON [espressione | colonna | alias | REPORT | ROW]...]
```

**DESCRIZIONE** *espressione* rappresenta una colonna o un'espressione. COMPUTE esegue calcoli su colonne o espressioni selezionate da una tabella. Funziona solamente con il comando BREAK.

Per default Oracle utilizza il nome della funzione (SUM, AVG e così via) come etichetta per il risultato nell'output della query. LABEL consente di specificare un *nome\_etichetta* che riscriva il valore predefinito.

OF specifica la colonna o l'espressione per la quale calcolare il valore. Queste colonne devono inoltre essere specificate nella clausola select, poiché, in caso contrario, il comando COMPUTE viene ignorato.

ON coordina i comandi COMPUTE e BREAK. COMPUTE stampa il valore calcolato e riavvia il calcolo quando il valore di ON espressione cambia oppure quando si verifica un'interruzione di riga o report. Vedere BREAK per i dettagli relativi alla coordinazione.

COMPUTE da solo visualizza i calcoli in corso.

AVG, MAXIMUM, MINIMUM, STD, SUM e VARIANCE funzionano con espressioni numeriche. MAXIMUM e MINIMUM funzionano anche con espressioni di caratteri, ma non con DATE. COUNT e NUMBER funzionano con qualsiasi tipo di espressione.

Tutti questi tipi di calcolo, fatta eccezione per NUMBER ignorano le righe con valori NULL:

---

AVG	Calcola il valore medio.
COUNT	Conta i valori non nulli.
MAXIMUM	Fornisce il valore massimo.
MINIMUM	Fornisce il valore minimo.
NUMBER	Conta tutte le righe restituite.
STD	Calcola la deviazione standard.
SUM	Calcola la somma dei valori non nulli.
VARIANCE	Calcola la varianza.

---

I calcoli successivi vengono semplicemente elencati in ordine senza virgolette, come nel caso seguente:

```
compute sum avg max of Importo Aliquota on report
```

Questa istruzione calcola la somma, la media e il valore massimo sia di Importo che Aliquota per l'intero report.

### **ESEMPIO**

Per eseguire il calcolo per ogni classificazione di Elemento e per l'intero report, si può scrivere:

```
break on Report on Industria skip 1
compute sum of Volume on Industria Report
```

## **COMPUTER REMOTO**

Un computer remoto è un qualsiasi computer di rete diverso da quello da cui si sta effettuando l'interrogazione.

## **COMUNICAZIONE, PROTOCOLLO**

Il protocollo di comunicazione è uno dei mezzi standard utilizzati per collegare tra loro due computer, in modo che possano condividere informazioni. I protocolli sono formati da diversi livelli sia software che hardware e possono collegare tra di loro computer sia omologhi che eterogenei.

## **CONCAT**

Vedere SET, ||.

## **CONDIZIONE**

Una condizione è un'espressione il cui valore viene valutato TRUE o FALSE, come in Eta > 65.

## **CONNECT (Forma 1)**

**VEDERE ANCHE** COMMIT, DISCONNECT, Capitolo 22.

### **SINTASSI**

```
CON[NECT] [{utente[/password] [@database] | /} [AS SYSOPER|SYSDBA]];
```

**DESCRIZIONE** Per utilizzare questo comando occorre trovarsi in SQL\*PLUS, anche se non è necessario essere collegati a Oracle (vedere DISCONNECT). CONNECT effettua il commit di qualsiasi modifica pendente, esegue la disconnessione da Oracle e ricollega l'utente con il nome specificato tramite *utente*. Se viene omessa la *password*, il sistema richiede di introdurla. La password introdotta in risposta al prompt di richiesta non viene visualizzata.

@*database* collega al database specificato, che può trovarsi sul proprio host, oppure su un altro computer connesso tramite Oracle Net.

## **CONNECT (forma 2, interno SQL)**

**VEDERE ANCHE** COMMIT, DECLARE DATABASE, Capitolo 22.

## SINTASSI

```
EXEC SQL CONNECT
{ :utente IDENTIFIED BY :password | :password_utente }
[AT { database | :variabile_host}]
[USING :stringa_collegamento]
[ALTER AUTHORIZATION :nuova_password
| IN {SYSDBA | SYSOPER } MODE ]
```

**DESCRIZIONE** CONNECT collega un programma che risiede sull'host a un database locale o remoto. Può essere utilizzato più di una volta per il collegamento a più database. *:password\_utente* rappresenta una variabile host che contiene il nome dell'utente di Oracle e la password separati da una barra (/). In alternativa, *:utente* e *:password* possono essere inserite separatamente utilizzando il secondo formato.

AT viene utilizzato per specificare un database diverso da quello predefinito attivo per l'utente in questione. Si tratta di una clausola richiesta per raggiungere qualunque database diverso da quello predefinito per l'utente. Questo nome può essere utilizzato successivamente in altre istruzioni SQL con AT. Questo database dev'essere prima identificato con DECLARE DATABASE. USING specifica una stringa di Oracle Net opzionale (quale il nome di un servizio) utilizzata durante l'operazione di connessione. Senza la stringa USING, la connessione avviene con il database predefinito dell'utente, a prescindere dal database specificato alla riga AT.

## CONNECT BY

**VEDERE ANCHE** Capitolo 13.

### SINTASSI

```
SELECT espressione [.espressione]...
  FROM [utente.]tabella
    WHERE condizione
    CONNECT BY [PRIOR] espressione = [PRIOR] espressione
      START WITH espressione = espressione
        ORDER BY espressione
```

**DESCRIZIONE** CONNECT BY è un operatore utilizzato in un'istruzione select per creare report sulle gerarchie di eredità in dati strutturati ad albero, come le organizzazioni di società, gli alberi genealogici e così via. START WITH specifica il punto dell'albero da cui iniziare. Le regole da seguire sono le seguenti:

- la posizione di PRIOR in relazione alle espressioni CONNECT BY determina quale espressione identifica la radice e quale identifica i rami dell'albero;
- una clausola where elimina elementi singoli dall'albero, ma non i discendenti (o progenitori, in base alla locazione di PRIOR);
- una specifica nella clausola CONNECT BY (in particolare un segno di diverso anziché di uguale) elimina sia un elemento singolo sia tutti i suoi discendenti;
- CONNECT BY non può essere utilizzato con un join di tabella nella clausola where.

### ESEMPIO

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sesso, Datanascita
  from ALLEVAMENTO
```

---

```
connect by Figlio = PRIOR Mucca
start with Figlio = 'DELLA'
order by Datanascita;
```

In questo esempio, la clausola seguente:

```
connect by Figlio = PRIOR Mucca
```

significa che figlio è la mucca precedente (PRIOR) a questa.

## **CONNETTERSI**

Connettersi significa identificarsi a Oracle tramite il proprio nome utente e la propria password allo scopo di accedere al database.

## **CONTAINS**

CONTAINS serve per valutare le ricerche di testo che usano gli indici CONTEXT all'interno di Oracle Text. Si *consulti* il Capitolo 24. Nella tabella seguente sono elencati gli operatori supportati delle ricerche di testo per CONTAINS.

---

OPERATORE	DESCRIZIONE
OR	Restituisce un record se almeno uno dei termini di ricerca ha un punteggio superiore alla soglia.
	Identico a OR.
AND	Restituisce un record se entrambi i termini di ricerca hanno un punteggio superiore alla soglia.
&	Identico a AND.
ACCUM	Restituisce un record se la somma dei punteggi dei termini di ricerca supera la soglia.
,	Identico a ACCUM.
MINUS	Restituisce un record se la differenza di punteggio fra il primo e il secondo termine di ricerca supera la soglia.
-	Identico a MINUS.
*	Assegna pesi differenti ai punteggi di ricerca.
NEAR	Il punteggio si baserà sulla vicinanza reciproca dei termini di ricerca nel testo esaminato.
:	Identico a NEAR.
{}	Racchiude parole riservate, come AND, se fanno parte di un termine di ricerca.
%	Carattere jolly per caratteri multipli.
_	Carattere jolly per un unico carattere.
\$	Esegue l'espansione etimologica del termine da cercare prima di effettuare la ricerca.
?	Effettua una corrispondenza non esatta (che trova anche gli errori di digitazione) del termine di ricerca prima di effettuare la ricerca vera e propria.
!	Esegue una ricerca SOUNDEX (fonetica).
()	Specifica l'ordine di valutazione dei criteri di ricerca.

---

## CONTEXT INDEX

Oracle Text supporta tre tipi di indici testuali; gli indici CONTEXT, gli indici CTXCAT e infine gli indici CTXRULE. Gli indici CONTEXT usano l'operatore CONTAINS e supportano una gamma più ampia di capacità per le ricerche di testo. Gli indici CTXCAT supportano una serie più limitata di operatori di ricerca ma supportano la creazione di gruppi di indici. Gli indici CTXRULE sono indici su colonne che contengono un gruppo di query di testo. Per ulteriori informazioni si consulti il Capitolo 24.

## CONTROL FILE (DATABASE)

Un control file è un piccolo file amministrativo richiesto da ciascun database, necessario per avviare ed eseguire un sistema di database. Un control file è abbinato a un database, non a un'istanza. Sono preferibili più control file identici rispetto a un unico file.

## CONTROL FILE (SQL\*LOADER)

Un control file di SQL\*Loader comunica a SQL\*Loader eseguibile dove trovare i dati da caricare e come elaborarli durante il caricamento. A ogni sessione SQL\*Loader è associato un control file. Per la sintassi dei control file, si consulti la voce SQLLDR. Per i dettagli sull'uso di SQL\*Loader, si rimanda al Capitolo 21.

## CONVERSIONE, FUNZIONI

**VEDERE ANCHE** CARATTERI, FUNZIONI; NUMERI, FUNZIONI; Capitolo 10.

**DESCRIZIONE** Di seguito è riportato un elenco in ordine alfabetico di tutte le funzioni di conversione e trasformazione attualmente disponibili in SQL di Oracle.

NOME FUNZIONE	DEFINIZIONE
ASCIISTR	Traduce una stringa in qualunque set di caratteri e restituisce una stringa ASCII nel set di caratteri del database.
BIN_TO_NUM	Converte un valore binario nel suo equivalente numerico.
CAST	Effettua una conversione CAST di un tipo incorporato o collection in un altro; utilizzato comunemente con tabelle annidate e array variabili.
CHARTOROWID	Abbreviazione di CHARacter TO ROW IDentifier. Modifica una stringa di caratteri in modo che funzioni come un identificatore di riga interno di Oracle o RowID.
COMPOSE	Traduce una stringa in qualunque tipo di dati in una stringa UNICODE nella sua forma completamente normalizzata utilizzando il medesimo set di caratteri della stringa di input.
CONVERT	Converte una stringa di caratteri dal set di caratteri di una lingua in un altro.
DECODE	Decodifica una stringa di caratteri CHAR o VARCHAR2 o un numero (NUMBER) in una serie di vari numeri, in base a un valore. Si tratta di una funzione di tipo <i>if, then, else</i> molto potente. A essa è dedicato il Capitolo 17.
DECOMPOSE	Traduce una stringa in qualunque tipo di dati in una stringa UNICODE dopo la sua scomposizione canonica utilizzando il medesimo set di caratteri della stringa di input.

(segue)

*(continua)*

NOME FUNZIONE	DEFINIZIONE
HEXTORAW	Abbreviazione di HEXadecimal TO RAW. Modifica una stringa di caratteri di numeri esadecimali in binario.
NUMTODSINTERVAL	Converte un numero in un letterale INTERVAL DAY TO SECOND.
NUMTOYMINTERVAL	Converte un numero in un letterale INTERVAL YEAR TO MONTH.
RAWTOHEX	Abbreviazione di RAW TO HEXadecimal. Modifica una stringa di numeri binari in una stringa di caratteri di numeri esadecimali.
RAWTONHEX	RAW TO NHEX. Converte un dato RAW in un valore NVARCHAR2 contenente l'equivalente esadecimale.
ROWIDTOCHAR	ROW Identifier TO CHARacter. Modifica un identificatore di riga interno di Oracle, o RowID, in una stringa di caratteri.
ROWIDTONCHAR	RAW TO NCHAR. Converte un valore RowID in tipo di dati NVARCHAR2.
TO_CHAR	TO CHARacter. Converte un NUMBER o DATE in una stringa di caratteri.
TO_CLOB	TO CLOB. Converte i valori NCLOB in una colonna LOB o in altre stringhe di caratteri in valori CLOB.
TO_DATE	TO DATE. Converte un elemento di tipo NUMBER, CHAR o VARCHAR2 in un elemento di tipo DATE (un tipo di dati di Oracle).
TO_DSINTERVAL	Converte una stringa di tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un tipo INTERVAL DAY TO SECOND.
TO_LOB	TO LOB. Converte un valore LONG in un LOB come parte di un'istruzione insert as select.
TO_MULTI_BYTE	TO MULTI_BYTE. Converte i caratteri a singolo byte di una stringa di caratteri in caratteri di più byte.
TO_NCHAR	TO NCHAR. Converte una stringa di caratteri, i numeri o le date dal set di caratteri del database in quello della lingua nazionale.
TO_NCLOB	TO NCLOB. Converte i valori CLOB in una colonna LOB o in altre stringhe di caratteri in valori NCLOB.
TO_NUMBER	TO NUMBER. Converte un CHAR o un VARCHAR2 in un NUMBER.
TO_SINGLE_BYTE	TO SINGLE BYTE. Converte i caratteri a più byte di un CHAR o un VARCHAR2 in byte singoli.
TO_YMINTERVAL	Converte una stringa di tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un tipo INTERVAL YEAR TO MONTH .
TRANSLATE	Traduce i caratteri contenuti in una stringa in caratteri differenti.
UNISTR	Converte una stringa Unicode nel set di caratteri Unicode del database.

## CONVERT

**VEDERE ANCHE** CONVERSIONE, FUNZIONI.

**SINTASSI**

CONVERT(*stringa*,[*set\_destinazione*,[*set\_sorgente*]])

**DESCRIZIONE** CONVERT converte i caratteri di una *stringa* da uno standard di rappresentazione a bit in un altro come, per esempio, da US7ASCII (il set predefinito se non ne vengono inseriti altri) a WE8DEC. Solitamente questa operazione viene eseguita quando i dati inseriti in una colonna su un computer contengono caratteri che non possono essere visualizzati o stampati correttamente su un altro computer. CONVERT consente di effettuare, nella maggior parte dei casi, una traduzione idonea da uno all'altro standard. I set di caratteri più comuni sono i seguenti:

F7DEC	Set a 7 bit ASCII di DEC per la Francia.
US7ASCII	Set US a 7-bit ASCII Standard.
WE8DEC	Set a 8-bit ASCII di DEC per l'Europa occidentale.
WE8HP	Set a 8-bit ASCII di HP per l'Europa occidentale.
WE8ISO8859P1	Set di caratteri a 8-bit ISO 8859-1 per l'Europa occidentale.
WE8EBCDIC500	IBM West European EBCDIC Code Page 500
WE8PC850	IBM PC Code Page 850

## COPY

**VEDERE ANCHE** CREATE DATABASE LINK, Capitolo 22.

### SINTASSI

```
COPY [FROM utente/password@database]
      [TO utente/password@database]
      {APPEND | CREATE | INSERT | REPLACE}
      table [ (colonna [.colonna]...) ]
      USING query
```

**DESCRIZIONE** COPY copia da una tabella in un'altra tabella che risiede su un computer differente, tramite Oracle Net. FROM specifica il nome utente, la password e il database della tabella di origine, mentre TO rappresenta la tabella di destinazione. Sia FROM che TO possono essere omessi, nel qual caso al posto della clausola mancante viene utilizzato il database predefinito dell'utente. Il database di origine e quello di destinazione non devono coincidere, perciò può essere omessa solo una delle clausole from e to.

APPEND aggiunge dati alla tabella di destinazione; se la tabella non esiste, viene creata. CREATE chiede di creare la tabella di destinazione; se questa esiste già, compare il messaggio d'errore ‘table already exists’, per informare l'utente che la tabella esiste già. INSERT inserisce dati nella tabella di destinazione; se la tabella non esiste compare il messaggio d'errore ‘table does not exist’, per informare l'utente che la tabella non esiste. REPLACE cancella i dati all'interno della tabella di destinazione e li sostituisce con i dati provenienti dalla tabella di origine; se la tabella non esiste, viene creata.

*tabella* è il nome della tabella di destinazione. *colonna* rappresenta il nome o i nomi delle colonne all'interno della tabella di destinazione. Se specificato, il numero delle colonne dev'essere uguale a quello delle colonne nella query. Se non viene specificata alcuna colonna, alle colonne copiate nella tabella di destinazione vengono assegnati gli stessi nomi delle colonne contenute nella tabella di origine. *query* identifica la tabella di origine e stabilisce quali righe e colonne copiare da quest'ultima.

**SET LONG** (*vedere SET*) determina la lunghezza di un campo LONG che può essere copiato. Le colonne lunghe con dati di lunghezza maggiore rispetto al valore di LONG vengono troncate. **SET COPYCOMMIT** determina quanti gruppi di righe copiare prima di effettuare un commit. **SET ARRAYSIZE** stabilisce quante righe possono essere contenute in un gruppo.

### ESEMPIO

Questo esempio copia i prestiti effettuati in una biblioteca dal database EDMESTON al database cui l'utente SQL\*PLUS locale è connesso. La tabella PRESTITO\_LOCALE viene creata dalla copia. Una volta giunte a destinazione le colonne possono essere rinominate. Si osservi l'uso del trattino (-) alla fine di ogni linea. Questo trattino è richiesto. Il comando non termina con un punto e virgola (poiché si tratta di un comando SQL\*Plus, e non di un comando SQL). Per le opzioni relative al comando COPY si consulti il comando SET.

```
copy from PRATICA/PRATICA@EDMESTON -
create PRESTITO_LOCALE (Prestito, Titolo) -
using select Nome, Titolo -
      from BIBLIOTECA_PRESTITO
```

## CORR

**VEDERE ANCHE** FUNZIONI DI GRUPPO.

### SINTASSI

```
CORR ( espr1 , espr2 ) [OVER ( clausola_analitica )]
```

**DESCRIZIONE** CORR restituisce il coefficiente di correlazione di un gruppo di coppie di numeri. *espr1* e *espr2* sono entrambe espressioni numeriche. Oracle applica la funzione al gruppo (*espr1*, *espr2*) dopo aver eliminato le coppie per le quali *espr1* o *espr2* è NULL. Quindi, Oracle esegue questo calcolo:

$$\text{COVAR\_POP}(\text{espr1}, \text{espr2}) / (\text{STDDEV\_POP}(\text{espr1}) * \text{STDDEV\_POP}(\text{espr2}))$$

La funzione restituisce un valore di tipo NUMBER. Se la funzione viene applicata a un gruppo vuoto, restituisce NULL.

## CORRISPONDENZA NON ESATTA

Una corrispondenza non esatta è un tipo di ricerca di testo che permette di ignorare eventuali errori di scrittura. *Vedere* CONTAINS e il Capitolo 24.

## COS

**VEDERE ANCHE** ACOS, ASIN, ATAN, ATAN2, COSH, EXP, LN, LOG, SIN, SINH, TAN, TANH.

### SINTASSI

```
COS(valore)
```

**DESCRIZIONE** COS restituisce il coseno di un valore, un angolo espresso in radianti. Per convertire in radianti la misura di un angolo espressa in gradi occorre moltiplicarla per pi/180.

## COSH

**VEDERE ANCHE** ACOS, ASIN, ATAN, ATAN2, COS, EXP, LN, LOG, SIN, SINH, TAN, TANH.

### SINTASSI

COSH(*valore*)

**DESCRIZIONE** COSH restituisce il coseno iperbolico di un *valore*.

## COUNT

**VEDERE ANCHE** FUNZIONI DI GRUPPO, Capitolo 8, Capitolo 11.

### SINTASSI

COUNT ( { \* | [ DISTINCT | ALL ] *expr* } ) [OVER ( *clausola\_analitica* )]

**DESCRIZIONE** COUNT conta il numero di righe in cui *espressione* non è NULL, che vengono poi restituite da una query. L'opzione DISTINCT fa in modo che vengano contate solo le righe non NULL distinte. Con il simbolo \*, vengono contate tutte le righe, sia NULL che non NULL.

## COVAR\_POP

**VEDERE ANCHE** FUNZIONI DI GRUPPO.

### SINTASSI

COVAR\_POP ( *expr1* , *expr2* ) [OVER ( *clausola\_analitica* )]

**DESCRIZIONE** COVAR\_POP restituisce la covarianza della popolazione di un gruppo di coppie di numeri. La si può utilizzare come un aggregato o come una funzione analitica.

*expr1* e *expr2* sono entrambe espressioni numeriche. Oracle applica la funzione al gruppo di coppie (*expr1* , *expr2*) dopo aver eliminato tutte le coppie per le quali *expr1* o *expr2* è NULL. Quindi, Oracle esegue questo calcolo:

$$(\text{SUM}(\text{expr1} * \text{expr2}) - \text{SUM}(\text{expr2}) * \text{SUM}(\text{expr1}) / n) / n$$

in cui n è il numero di coppie (*expr1* , *expr2*) dove né *expr1* né *expr2* è NULL.

## COVAR\_SAMP

**VEDERE ANCHE** FUNZIONI DI GRUPPO.

### SINTASSI

COVAR\_POP ( *expr1* , *expr2* ) [OVER ( *clausola\_analitica* )]

**DESCRIZIONE** COVAR\_SAMP restituisce la covarianza del campione per un gruppo di coppie di numeri. La si può utilizzare come un aggregato o come una funzione analitica.

*expr1* e *expr2* sono entrambe espressioni numeriche. Oracle applica la funzione al gruppo di coppie (*expr1* , *expr2*) dopo aver eliminato tutte le coppie per le quali *expr1* o *expr2* è NULL. Quindi, Oracle esegue questo calcolo:

$(\text{SUM}(\text{espr1} * \text{espr2}) - \text{SUM}(\text{espr1}) * \text{SUM}(\text{espr2}) / n) / (n-1)$

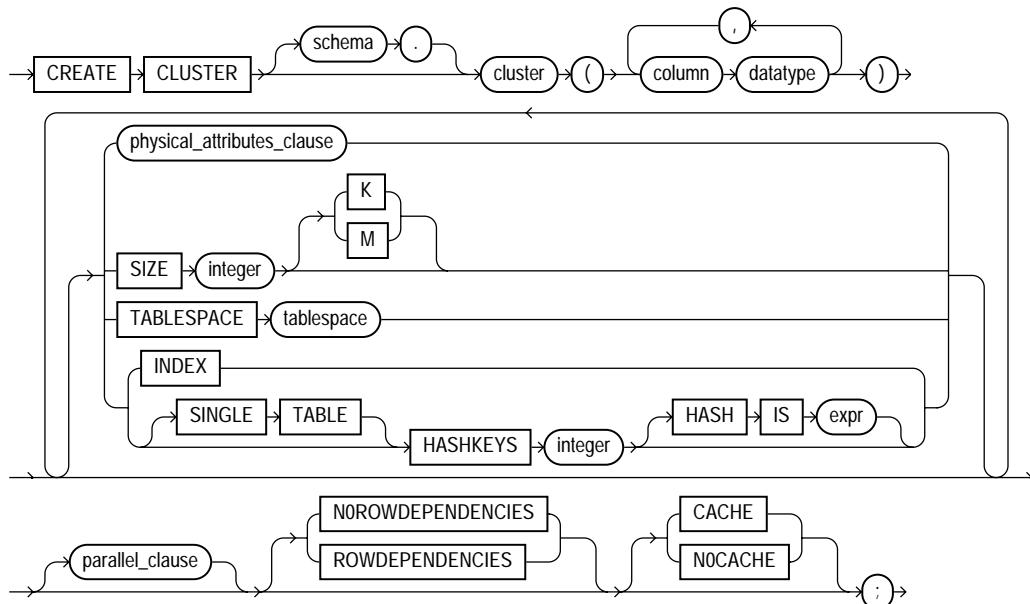
in cui n è il numero di copie (*espr1*, *espr2*) dove né *espr1* né *espr2* è **NULL**.

## CREATE CLUSTER

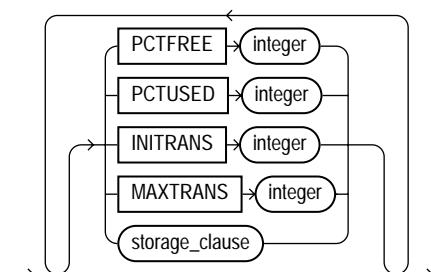
**VEDERE ANCHE** CREATE INDEX, CREATE TABLE, Capitolo 20.

### SINTASSI

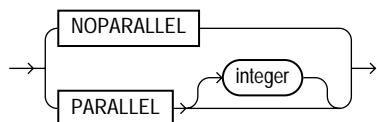
create\_cluster



### physical\_attributes\_clause



### parallel\_clause



**DESCRIZIONE** CREATE CLUSTER crea un cluster per una o più tabelle. Le tabelle vengono aggiunte al cluster tramite CREATE TABLE con la clausola cluster. CREATE CLUSTER richiede almeno una colonna cluster da ciascuna delle tabelle. Queste devono avere la stessa dimensione ed essere dello stesso tipo di dati, ma non è richiesto che siano omonime. Per le tabelle di un cluster, le righe con gli stessi valori della colonna cluster vengono riunite insieme nella stessa area del disco, nello stesso blocco o negli stessi blocchi logici. In questo modo, le prestazioni possono migliorare quando le colonne cluster sono le colonne attraverso cui le tabelle vengono solitamente unite.

Ogni valore distinto in ogni colonna cluster viene memorizzato solo una volta, indipendentemente dal fatto che ricorra più volte nelle tabelle e nelle righe. Solitamente, in questo modo si riduce la quantità di spazio su disco necessaria per memorizzare le tabelle, ma ciascuna tabella continua ad apparire come se contenesse tutti i suoi dati. Le tabelle con colonne di tipo LONG non possono essere raggruppate in cluster.

*cluster* è il nome assegnato al cluster. *colonna* e *tipodato* seguono il metodo di CREATE TABLE, con l'unica eccezione dell'impossibilità di specificare NULL e NOT NULL. Tuttavia nell'istruzione CREATE TABLE vera e propria almeno una colonna di un cluster dev'essere NOT NULL. SIZE imposta la dimensione in byte di un blocco logico (diverso dal blocco fisico). SPACE rappresenta l'allocazione iniziale su disco del cluster, come viene utilizzata in CREATE TABLE.

SIZE dovrebbe essere la quantità media di spazio necessaria per memorizzare tutte le righe da tutte le tabelle contenute in cluster che risultano associate a un'unica chiave di clusterizzazione. Un valore di SIZE piccolo può far lievitare il tempo necessario per accedere alle tabelle nel cluster, ma può ridurre l'uso di spazio su disco. SIZE dovrebbe essere un divisore adeguato della dimensione di un blocco fisico. In caso contrario, Oracle utilizza il divisore successivo di valore maggiore. Se SIZE supera la dimensione di un blocco fisico, Oracle utilizza al suo posto la dimensione di un blocco fisico.

Per default, il cluster è indicizzato ed è necessario creare un indice sulla chiave di clusterizzazione prima di inserire i dati nel cluster. Se si specifica la forma di cluster hash, tuttavia, non è necessario (e non è nemmeno possibile) creare un indice sulla chiave di clusterizzazione. Al contrario, Oracle utilizza una funzione di hash per memorizzare le righe della tabella.

È possibile creare il proprio valore di hash come una colonna della tabella e utilizzarlo con la clausola HASH IS per dire a Oracle di utilizzare questa colonna come valore di hash. In caso contrario, Oracle utilizza una funzione di hash interna che si basa sulle colonne della chiave di clusterizzazione. La clausola HASHKEYS in realtà crea il cluster hash e specifica il numero di valori hash, arrotondati al numero primo più vicino. Il valore minimo è 2.

Per i dettagli sui parametri comuni della clausola di memorizzazione si consulti la voce STORAGE.

## CREATE CONTEXT

**VEDERE ANCHE** ALTER CONTEXT.

**SINTASSI**

```
CREATE [OR REPLACE] CONTEXT SPAZIO DI NOMI USING [SCHEMA .] PACKAGE
[ INITIALIZED { EXTERNALLY | GLOBALLY }
| ACCESSED GLOBALLY ] :
```

**DESCRIZIONE** Un contesto è un gruppo di attributi utilizzati per proteggere un'applicazione. CREATE CONTEXT crea uno spazio di nomi per un contesto e lo associa al package creato

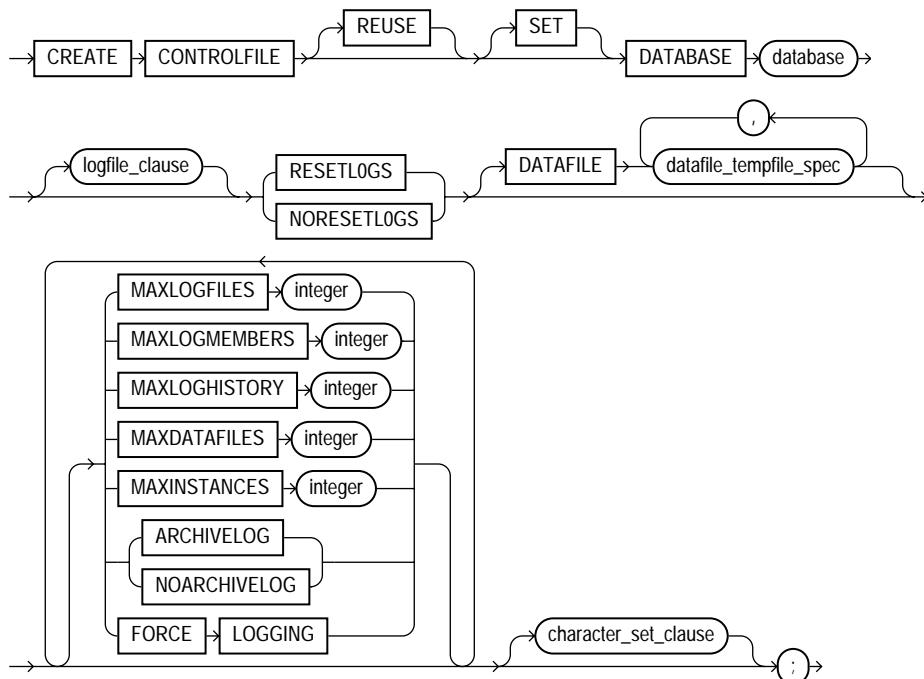
esternamente che imposta il contesto. Per creare lo spazio di nomi di un contesto è necessario disporre del privilegio di sistema CREATE ANY CONTEXT.

## CREATE CONTROLFILE

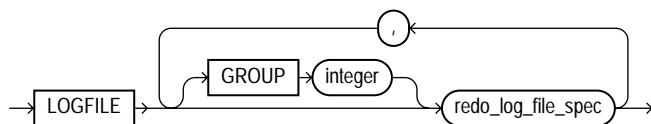
**VEDERE ANCHE** ALTER DATABASE, CREATE DATABASE.

### SINTASSI

`create_controlfile`



`logfile_clause`



`character_set_clause`



**DESCRIZIONE** Il comando CREATE CONTROLFILE ricrea un control file nel caso in cui l'utente abbia perso il proprio control file corrente a causa di un inconveniente tecnico, oppure nel caso si desideri modificare il nome del database o una delle opzioni per il file di log di un file di dati. In generale, questo comando dovrebbe essere utilizzato solo da amministratori di database esperti.

**NOTA** Si consiglia vivamente di eseguire un backup fuori linea completo di tutti i file di database prima di usare questo comando.

L'opzione REUSE consente di riutilizzare i control file esistenti anziché visualizzare un errore. L'opzione SET modifica il nome del database, specificato dalla clausola del database. La clausola LOGFILE specifica i gruppi di redo log file, ognuno dei quali deve esistere. La clausola RESETLOGS e la sua controparte NORESETLOGS comunicano, rispettivamente, a Oracle di azzerare i log correnti oppure no. La linea DATAFILE specifica i file di dati del database, ciascuno dei quali deve esistere.

L'opzione **MAXLOGFILES** specifica il numero massimo di gruppi di redo log file che possono essere creati. L'opzione **MAXLOGMEMBERS** specifica il numero di copie per un gruppo di redo log. L'opzione **MAXLOGHISTORY** specifica il numero di gruppi di redo log file archiviati per Real Application Clusters. L'opzione **MAXDATAFILES** specifica il numero massimo di file di dati che possono essere creati per il database. L'opzione **MAXINSTANCES** specifica il numero massimo di istanze di Oracle che possono eseguire il MOUNT del database e aprire il database. Le opzioni **ARCHIVELOG** e **NOARCHIVELOG** abilitano e disabilitano rispettivamente l'archiviazione dei redo log file.

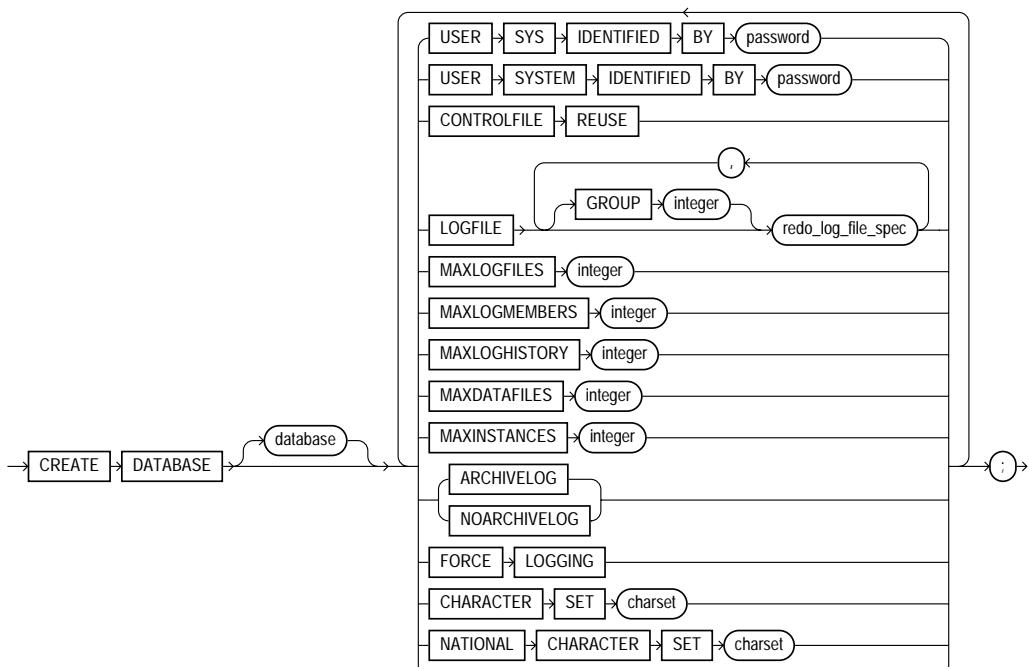
Il comando **CREATE CONTROLFILE** necessario per un database esistente, può essere generato attraverso il comando **ALTER DATABASE BACKUP CONTROLFILE TO TRACE**.

## **CREATE DATABASE**

**VEDERE ANCHE** ALTER DATABASE, CREATE CONTROLFILE, CREATE ROLLBACK SEGMENT, CREATE TABLESPACE, SHUTDOWN, STARTUP, Capitolo 40.

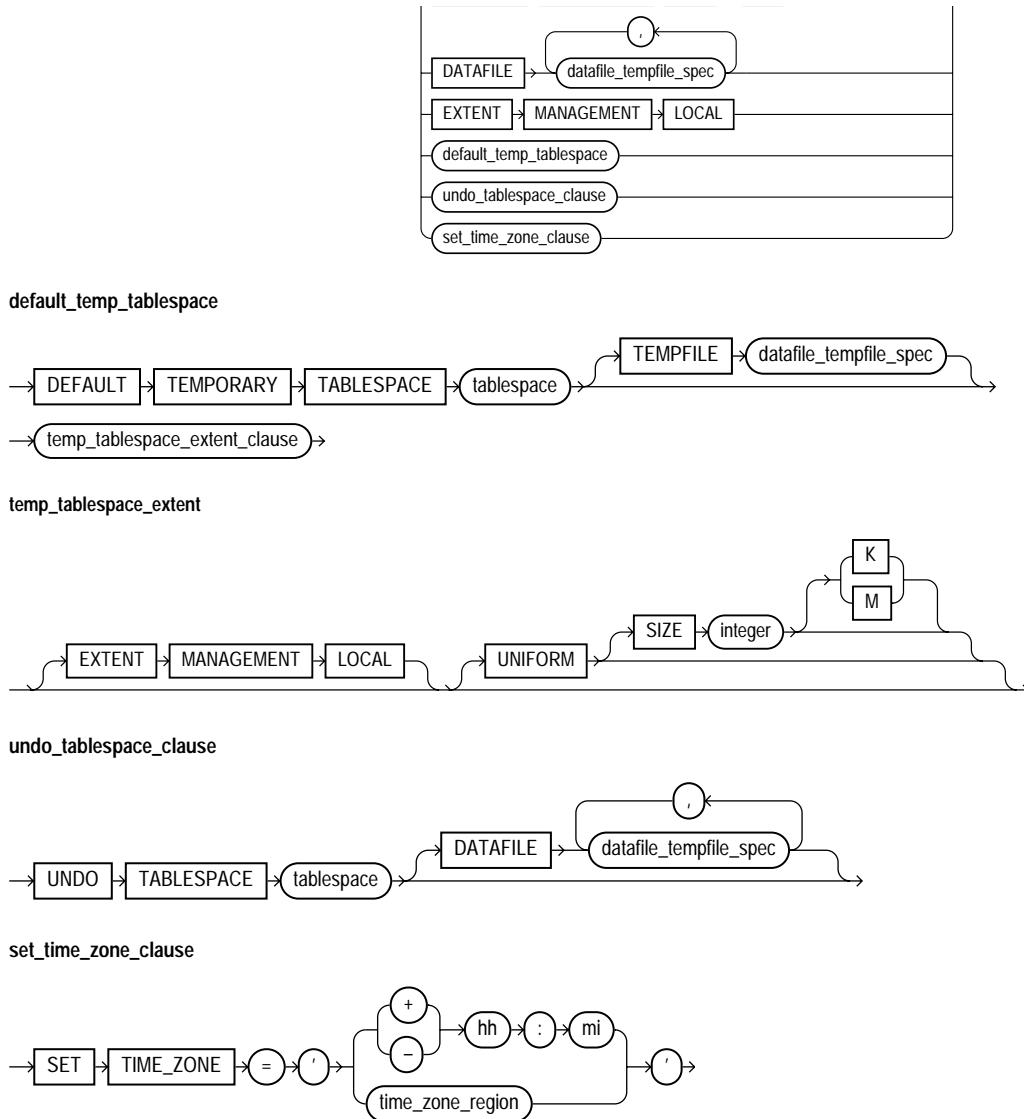
SINTASSI

create database



(segue)

(continua)



**DESCRIZIONE** *database* è il nome del database, e deve avere una lunghezza pari o inferiore a otto caratteri. DB\_NAME in init.ora contiene il nome di database predefinito. In generale, questo comando dovrebbe essere utilizzato solo da amministratori di database esperti.

**NOTA** L'uso di questo comando in un database già esistente comporta la cancellazione dei file di dati specificati.

*definizione\_file* specifica i nomi e le dimensioni dei file di dati e dei file di log:

'file' [SIZE *intero* [K | M] [REUSE]

**SIZE** rappresenta il numero di byte riservati per questo file. Il suffisso K moltiplica il valore specificato per 1024; M moltiplica il valore per 1048576. **REUSE** (senza **SIZE**) provoca la distruzione dei contenuti di qualsiasi file con quel nome e associa il nome al database in questione. **SIZE** con **REUSE** crea il file nel caso non esista ancora, altrimenti ne controlla la dimensione. **CONTROLFILE REUSE** sovrascrive i control file già esistenti definiti dal parametro **CONTROL\_FILES** nel file dei parametri di inizializzazione.

**LOGFILE** assegna il nome dei file da utilizzare come redo log file. Se questo parametro non viene utilizzato, per default Oracle ne crea due. **MAXLOGFILES** sostituisce il parametro di inizializzazione **LOG\_FILES** e definisce il numero massimo di redo log file che possono essere creati per il database. Questo numero non potrà essere aumentato in futuro se non quando si ricreerà il control file. Il valore minimo è 2. Un numero maggiore crea semplicemente un control file più grande.

**DATAFILE** specifica i nomi dei file che il database stesso utilizzerà. **MAXDATAFILES** impone il limite superiore assoluto per i file che possono essere creati per questo database e sostituisce il parametro di inizializzazione **DB\_FILES**. Un numero maggiore crea semplicemente un control file più grande.

Quando l'opzione **AUTOEXTEND** viene abilitata (ON) per un file di dati, il file di dati si estenderà dinamicamente in base alle necessità con incrementi di dimensione pari a **NEXT**, fino a un valore massimo pari a **MAXSIZE** (o **UNLIMITED**, illimitato).

**MAXINSTANCES** sostituisce il parametro **INSTANCES** in init.ora e impone il numero massimo di istanze simultanee che possono eseguire il **MOUNT** del database e aprire il database.

**ARCHIVELOG** e **NOARCHIVELOG** definiscono il modo in cui i redo log file vengono utilizzati quando il database viene creato per la prima volta. **NOARCHIVELOG** è l'impostazione predefinita e significa che i file di redo vengono riutilizzati senza salvarne il contenuto altrove. In questo modo, viene garantito il recupero dell'istanza, ma non il recupero dei dati a fronte di un malfunzionamento del sistema, quale un crash del disco. **ARCHIVELOG** impone l'archiviazione dei file di redo (solitamente su un altro disco o nastro), in modo che sia possibile effettuare il ripristino dei dati a fronte di un malfunzionamento del sistema. Questa modalità gestisce anche il ripristino dell'istanza. Questo parametro può essere reimpostato tramite il comando **ALTER DATABASE**.

L'opzione **MAXLOGMEMBERS** specifica il numero massimo di copie di un gruppo di redo log file. L'opzione **MAXLOGHISTORY** specifica il numero massimo di redo log file archiviati, utile solo per Real Application Cluster quando si archiviano i file di log. L'opzione **CHARACTER SET** specifica l'insieme di caratteri utilizzato per memorizzare i dati, che dipende dal sistema operativo.

Per una gestione maggiormente automatizzata dei segmenti (di rollback) di undo è possibile specificare la clausola **UNDO TABLESPACE** per allocare una tablespace appositamente per contenere i dati degli undo. Il database dev'essere avviato nella modalità Automatic Undo Management (AUM).

La clausola **DEFAULT TEMPORARY TABLESPACE** può essere utilizzata per designare una tablespace non **SYSTEM** come tablespace temporanea predefinita per tutti i nuovi utenti creati nel database.

## CREATE DATABASE LINK

**VEDERE ANCHE** CREATE SYNONYM, SELECT, Capitolo 22.

## SINTASSI

```
CREATE [SHARED] [PUBLIC] DATABASE LINK dblink
[ CONNECT TO { CURRENT_USER | utente IDENTIFIED BY password
[AUTHENTICATED BY utente IDENTIFIED BY password] }
| AUTHENTICATED BY utente IDENTIFIED BY password]
[USING 'stringa_connessione'];
```

**DESCRIZIONE** *dblink* è il nome assegnato al collegamento. *stringa\_connessione* è la definizione del database remoto cui è possibile accedere tramite Oracle Net e definisce il collegamento tra un database locale e il nome di un utente su un database remoto. I collegamenti PUBLIC possono essere creati solamente da un utente che dispone del privilegio di sistema CREATE PUBLIC DATABASE LINK, ma sono accessibili a tutti gli utenti eccetto quelli che hanno creato un collegamento privato con lo stesso nome. Se non si specifica PUBLIC, il collegamento è disponibile solo per l'utente che ha eseguito l'istruzione CREATE DATABASE LINK. *stringa\_connessione* è il nome di servizio impiegato da Oracle Net per il database remoto.

L'accesso alle tabelle remote è analogo all'accesso alle tabelle locali, l'unica eccezione è che il nome della tabella remota dev'essere seguito da *@link* nella clausola from dell'istruzione select. La maggior parte dei sistemi imposta il numero massimo di collegamenti simultanei a quattro. Il DBA può incrementare questo numero tramite il parametro OPEN\_LINKS di init.ora.

Le query con struttura ad albero sono limitate. Possono anche non utilizzare l'operatore PRIOR ma devono farlo nella clausola connect by. START WITH non può contenere una subquery. CONNECT BY e START WITH non possono utilizzare la funzione USERENV('ENTRYID') o la pseudocolonna RowNum.

Per creare un database link, è necessario il privilegio CREATE DATABASE LINK nel database locale e il privilegio CREATE SESSION in un database remoto. Per creare un database link pubblico, è necessario disporre del privilegio di sistema CREATE PUBLIC DATABASE LINK.

Se si usa la clausola CONNECT TO CURRENT\_USER, il collegamento cercherà di aprire una connessione nel database remoto con il nome utente e la password correnti dell'utente. Per questo motivo, si dovrà coordinare qualsiasi modifica di password effettuata tra il database locale e il database remoto altrimenti i database link potrebbero smettere di funzionare.

Se si utilizza l'architettura a server condiviso, è possibile creare database link SHARED che eliminano la necessità di più connessioni dedicate separate tramite collegamenti. Quando si crea un collegamento SHARED, è necessario fornire un nome utente valido e una password valida nel database remoto da utilizzare come autenticazione per la connessione.

## ESEMPI

Quanto segue definisce un collegamento di nome EDMESTON\_LIBRI che connette al nome utente Pratica nel database EDMESTON:

```
create database link EDMESTON_LIBRI
connect to Pratica identified by Pratica
using 'EDMESTON';
```

A questo punto, si può eseguire una query sulle tabelle di Pratica, come mostrato di seguito:

```
select Titolo, Editore
from BIBLIOTECA@EDMESTON_LIBRI;
```

Inoltre, si potrebbe creare un sinonimo per celare il fatto che le tabelle sono remote:

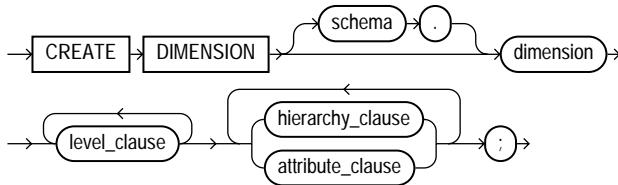
```
create synonym BIBLIOTECA for BIBLIOTECA@EDMESTON_LIBRI;
```

## CREATE DIMENSION

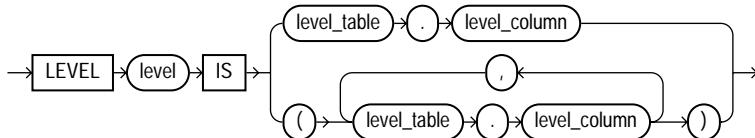
**VEDERE ANCHE** ALTER DIMENSION, DROP DIMENSION.

### SINTASSI

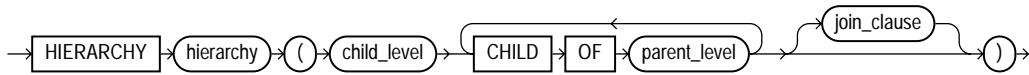
`create_dimension`



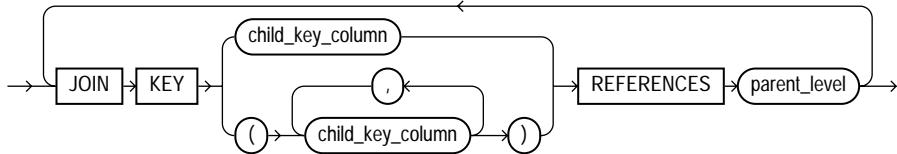
`level_clause`



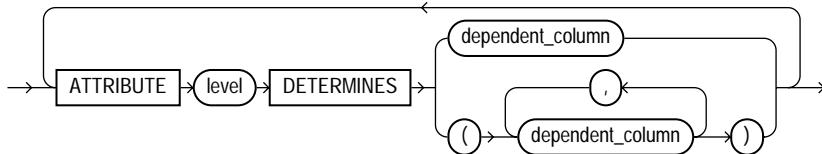
`hierarchy_clause`



`join_clause`



`attribute_clause`



**DESCRIZIONE** CREATE DIMENSION crea delle gerarchie tra le colonne correlate nelle tabelle in modo che possano essere utilizzate dall'ottimizzatore. L'ottimizzatore usa i valori della dimensione quando stabilisce se una vista materializzata restituisce o meno gli stessi dati della sua tabella base. Per creare una dimensione, è necessario disporre del privilegio CREATE DIMENSION; per creare una dimensione nello schema di un altro utente si deve possedere il privilegio CREATE ANY DIMENSION.

LEVEL definisce il livello all'interno della dimensione. HIERARCHY definisce le relazioni tra i livelli. ATTRIBUTE assegna attributi specifici ai livelli all'interno della dimensione. JOIN\_KEY definisce le clausole di join tra i livelli.

## ESEMPI

Per una tabella di nome PAESE con colonne Paese e Continente e per una seconda tabella denominata CONTINENTE con una colonna Continente:

```
create table CONTINENTE (
    Continente      VARCHAR2(30));

create table PAESE (
    Paese          VARCHAR2(30) not null,
    Continente     VARCHAR2(30));

create dimension GEOGRAFIA
    level PAESE_ID      is PAESE.Paese
    level CONTINENTE_ID  is CONTINENTE.Continente
    hierarchy PAESE_ROLLUP (
        PAESE_ID           child of
        CONTINENTE_ID
    join key PAESE.Continente references CONTINENTE_ID);
```

## CREATE DIRECTORY

**VEDERE ANCHE** BFILE, Capitoli 25 e 30.

### SINTASSI

```
CREATE [OR REPLACE] DIRECTORY directory AS 'nome_percorso';
```

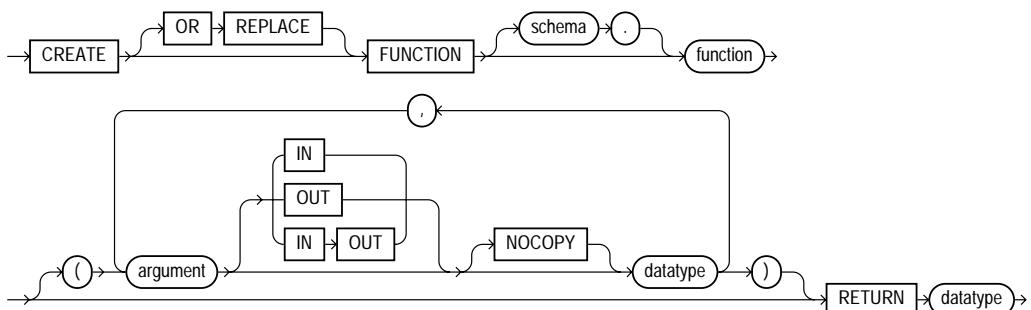
**DESCRIZIONE** In Oracle, una “directory” rappresenta un alias di una directory del sistema operativo. Prima di accedere ai valori del tipo di dati BFILE, è necessario creare una directory. Per dettagli sulla creazione e la gestione delle tabelle esterne, si rimanda al Capitolo 25.

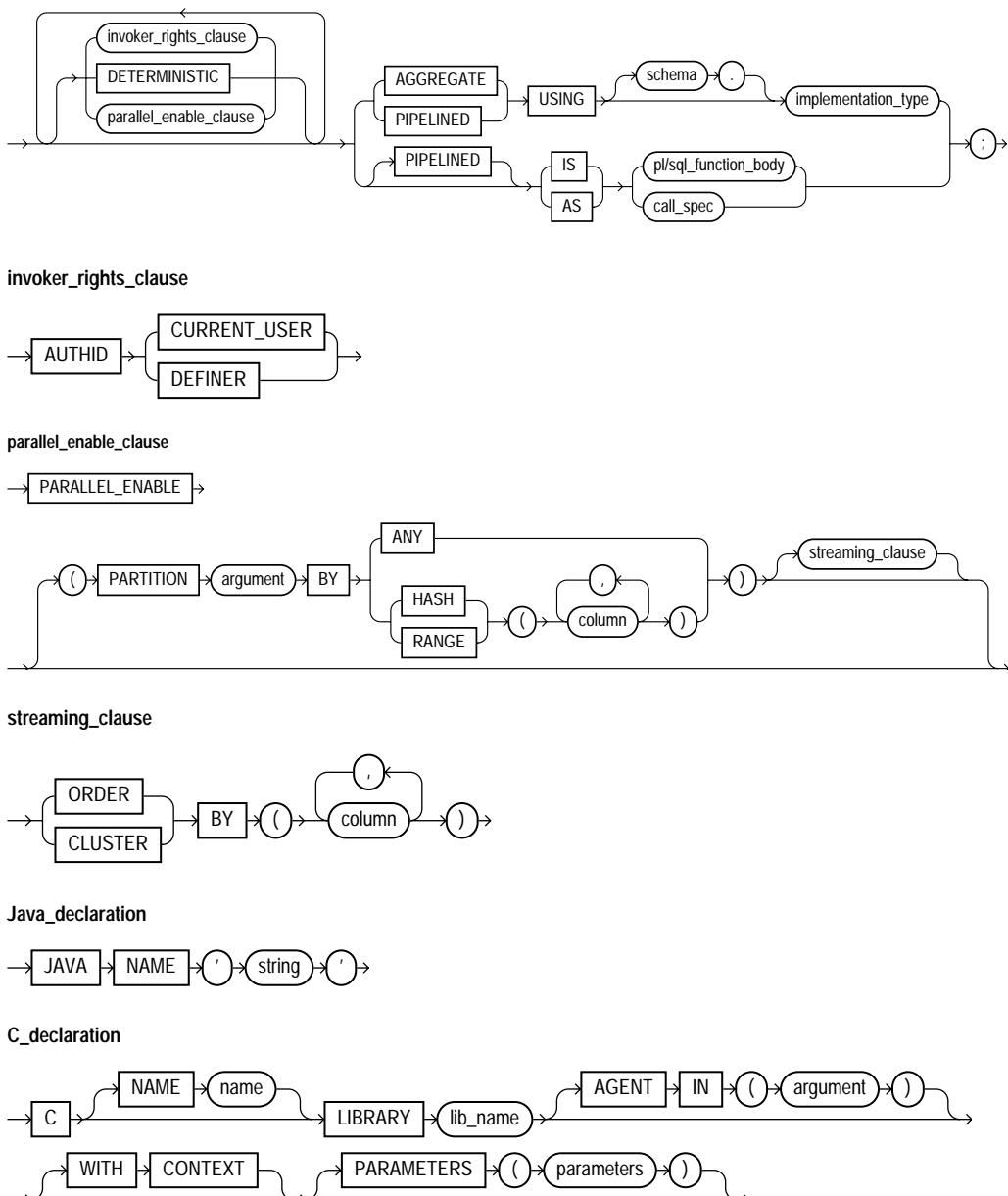
## CREATE FUNCTION

**VEDERE ANCHE** ALTER FUNCTION; BLOCCO, STRUTTURA; CREATE LIBRARY; CREATE PACKAGE; CREATE PROCEDURE; DATI, TIPI; DROP FUNCTION; Capitoli 27, 29 e 36.

### SINTASSI

create\_function





**DESCRIZIONE** *funzione* è il nome della funzione da definire. Una funzione può avere parametri, argomenti di un certo tipo di dati cui è stato assegnato un nome, e ciascuna funzione restituisce un valore di un certo tipo di dati come specificato dalla clausola RETURN. Il blocco PL/SQL definisce il comportamento della funzione come una serie di dichiarazioni, istruzioni di programma PL/SQL ed eccezioni.

Il qualificatore **IN** richiede di specificare un valore per il parametro nel momento in cui la funzione viene chiamata, ma poiché questa specifica va sempre fatta per le funzioni, la sintassi è facoltativa. In una procedura si possono avere altri tipi di parametri. La differenza tra una funzione e una procedura è che una funzione restituisce un valore all'ambiente chiamante.

Per creare una funzione, è necessario disporre del privilegio di sistema **CREATE PROCEDURE**. Per creare una funzione nell'account di un altro utente, occorre possedere il privilegio di sistema **CREATE ANY PROCEDURE**.

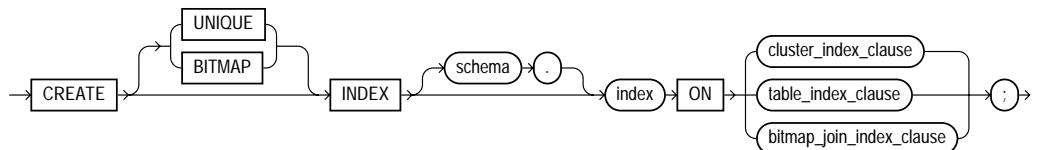
La funzione può servirsi delle librerie C memorizzate al di fuori del database (si consulti **CREATE LIBRARY**). Se all'interno della funzione si usa Java, è possibile fornire una dichiarazione Java all'interno della clausola **LANGUAGE**. La clausola **diritti\_chiamante** consente di specificare se la funzione viene eseguita con i privilegi del proprietario della funzione stessa oppure con i privilegi dell'utente corrente (l'utente chiamante).

## CREATE INDEX

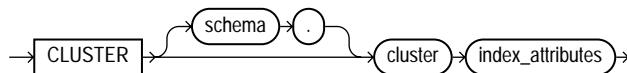
**VEDERE ANCHE** ANALYZE, ALTER INDEX, DROP INDEX, VINCOLO DI INTEGRITÀ, STORAGE, Capitoli 18 e 20.

### SINTASSI

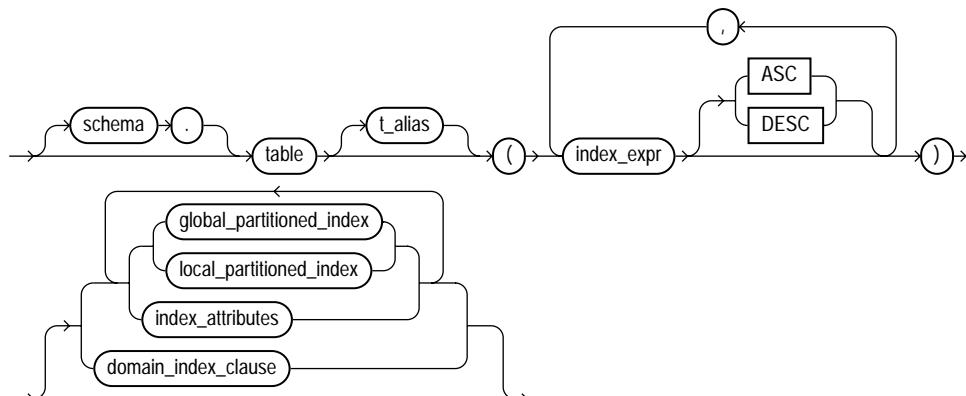
`create_index`

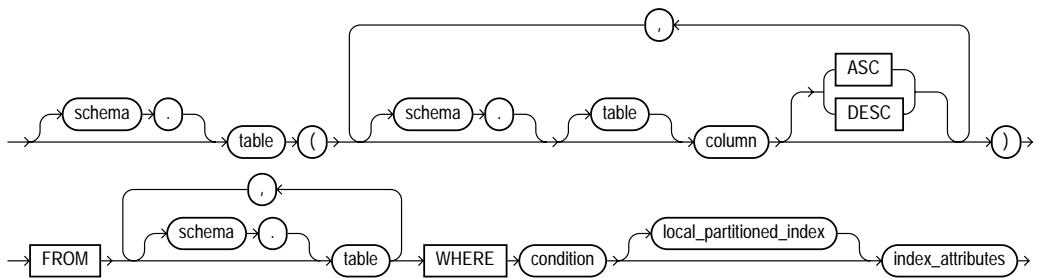
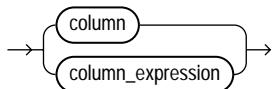
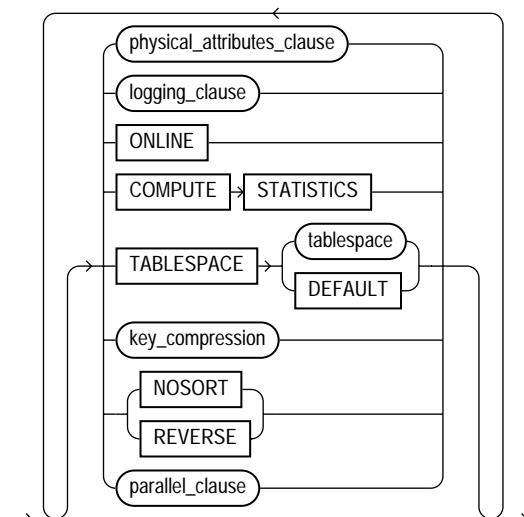
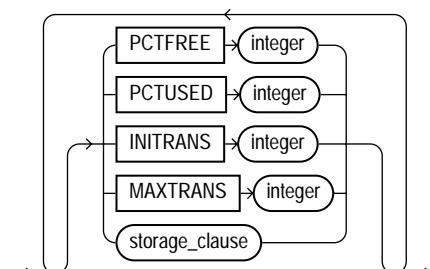


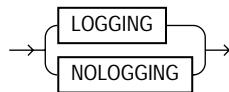
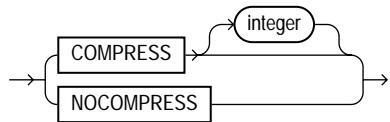
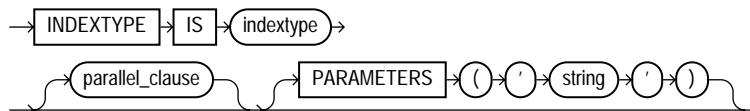
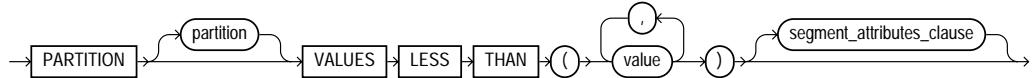
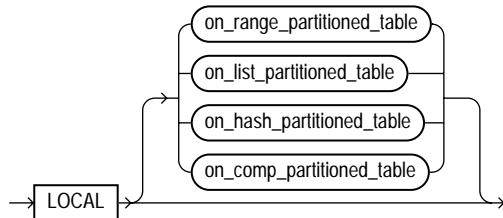
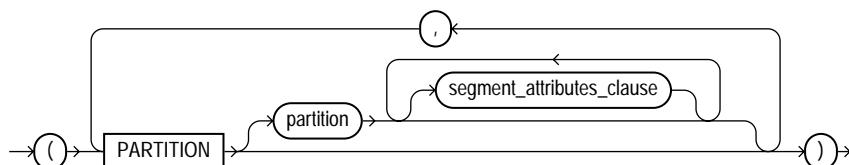
`cluster_index_clause`



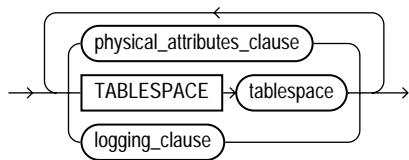
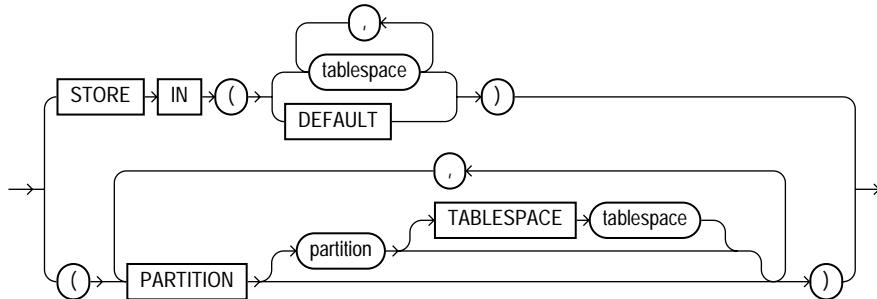
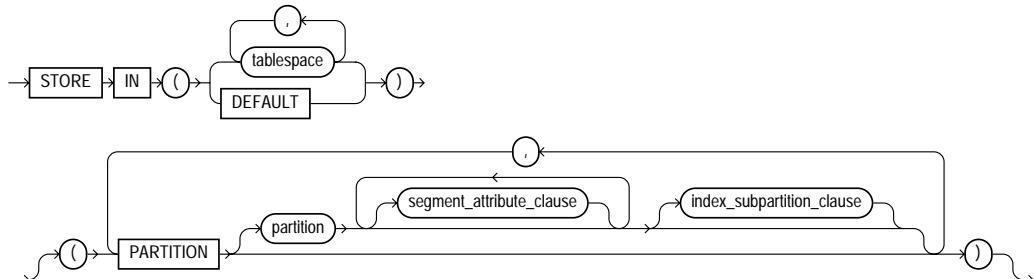
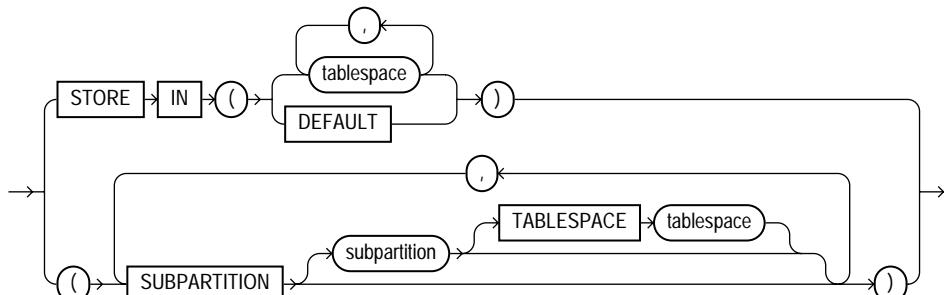
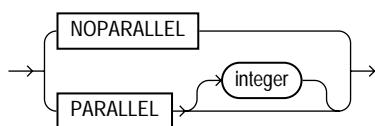
`table_index_clause`



**bitmap\_join\_index\_clause****index\_expr****index\_attributes****physical\_attributes\_clause**

**logging\_clause****key\_compression****domain\_index\_clause****global\_partitioned\_index****global\_partitioning\_clause****local\_partitioned\_index****on\_range\_partitioned\_table****on\_list\_partitioned\_table\_clause\_private**

→ This is identical to range partition →

**segment\_attributes\_clause****on\_hash\_partitioned\_table****on\_comp\_partitioned\_table****index\_subpartition\_clause****parallel\_clause**

**DESCRIZIONE** *indice* è il nome assegnato a questo indice. Solitamente è buona norma fare in modo che rispecchi i nomi delle colonne e della tabella da indicizzare. *tabella* e *colonna(e)* rappresentano la tabella e le colonne su cui creare l'indice. Un indice UNIQUE garantisce che ciascuna riga indicizzata sia unica per quanto riguarda i valori delle colonne dell'indice. Per creare automaticamente indici unici, si può utilizzare il vincolo unico sulle colonne. Se si specificano più colonne viene creato un indice composto. ASC e DESC significano rispettivamente crescente e decrescente; gli indici decrescenti sono supportati a partire da Oracle8i. CLUSTER è il nome della chiave di clusterizzazione che risulta indicizzata per un cluster. Le chiavi dei cluster devono essere indicizzate per consentire l'accesso alle tabelle loro associate (*vedere* CREATE TABLE per una descrizione di INITTRANS e MAXTRANS). Il valore predefinito di INITTRANS per gli indici è 2; MAXTRANS è 255. PCTFREE rappresenta la percentuale di spazio da lasciare libera nell'indice per i nuovi inserimenti e aggiornamenti. Il valore minimo è zero.

TABLESPACE è il nome della tablespace cui viene assegnato questo indice. La sezione degli attributi fisici contiene sottoclausole descritte in STORAGE. NOSORT è un'opzione il cui valore primario consiste nel ridurre il tempo di creazione di un indice, solo ed esclusivamente nel caso in cui i valori nella colonna da indicizzare siano già in ordine crescente. Non viene compromesso nulla se in seguito l'ordine crescente decade, tuttavia NOSORT funziona solo se questi valori sono in ordine quando viene creato l'indice. Se le righe non sono in ordine, CREATE INDEX restituisce un messaggio d'errore; non viene danneggiato nulla, ma viene consentito di eseguire di nuovo il comando senza l'opzione NOSORT.

PARALLEL, insieme a DEGREE e INSTANCES, specifica le caratteristiche parallele dell'indice. DEGREE specifica il numero dei server di query da utilizzare per creare l'indice; INSTANCES specifica in che modo l'indice dovrà essere suddiviso tra le istanze di Real Application Clusters per l'elaborazione in parallelo delle query. Un intero *n* specifica che l'indice deve essere suddiviso tra il numero specificato di istanze disponibili.

Per creare un indice, è necessario essere il proprietario della tabella indicizzata, disporre del privilegio INDEX sulla tabella oppure del privilegio di sistema CREATE ANY INDEX. Per creare un indice basato sulle funzioni, occorre possedere il privilegio QUERY REWRITE.

BITMAP crea un indice bitmap, che può essere utile per colonne con pochi valori distinti. Le clausole PARTITION creano indici sulle tabelle partizionate.

REVERSE memorizza i byte del valore indicizzato in ordine inverso. Non è possibile invertire un indice bitmap.

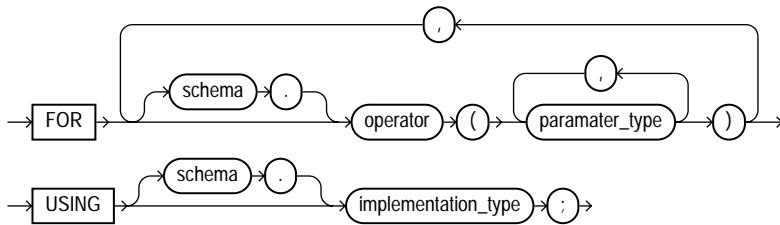
COMPRESS risparmia spazio di memorizzazione comprimendo indici non partizionati non unici. Durante le operazioni di recupero dei dati, questi verranno visualizzati come se non fossero compressi. È possibile disattivare la compressione dell'indice; ricreare l'indice con la clausola NOCOMPRESS.

## CREATE INDEXTYPE

### SINTASSI

create\_indextype





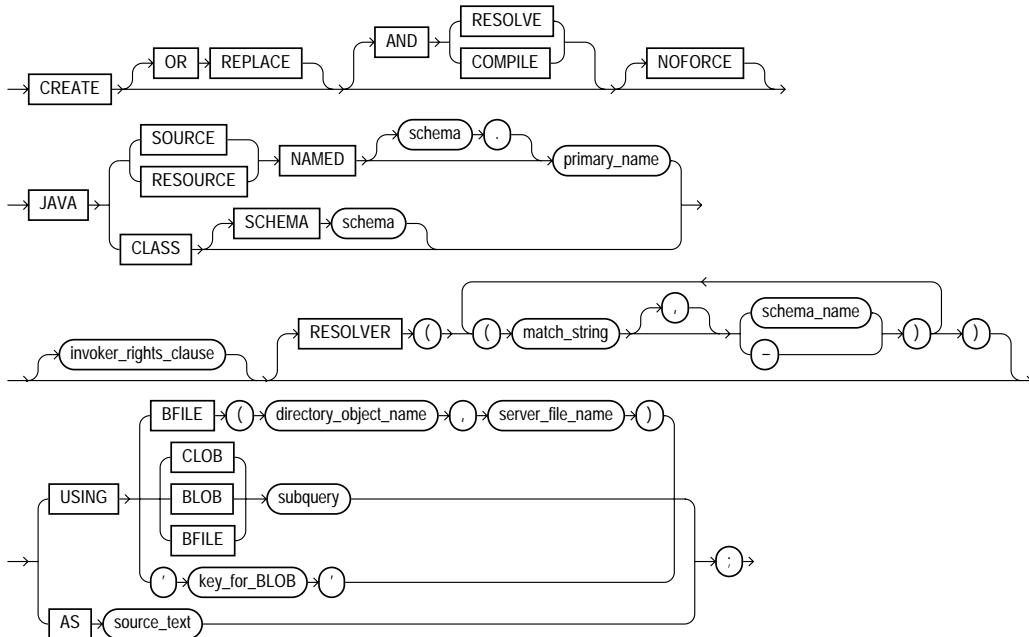
**DESCRIZIONE** `CREATE INDEXTYPE` specifica le routine impiegate da un indice di dominio. Per creare un tipo di indice, si deve disporre del privilegio di sistema `CREATE INDEXTYPE`. Per creare un tipo di indice nello schema di un altro utente, è necessario disporre dei privilegi di sistema `CREATE ANY INDEXTYPE`.

## CREATE JAVA

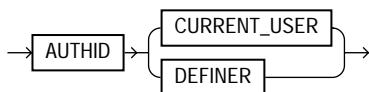
**VEDERE ANCHE** `ALTER JAVA`, `DROP JAVA`, Capitoli 34, 35 e 36.

### SINTASSI

`create_java`



`invoker_rights_clause`



**DESCRIZIONE** CREATE JAVA crea una sorgente, una classe o una risorsa Java. Si deve possedere il privilegio di sistema CREATE PROCEDURE oppure (per creare l'oggetto nello schema di un altro utente) CREATE ANY PROCEDURE. Per sostituire un oggetto dello schema simile nello schema di un altro utente, è necessario disporre del privilegio di sistema ALTER ANY PROCEDURE.

Le clausole JAVA SOURCE, JAVA CLASS e JAVA RESOURCE caricoano sorgenti, classi e risorse.

AUTHID consente di specificare se la funzione viene eseguita con i privilegi del proprietario della funzione o con quelli dell'utente corrente.

### ESEMPIO

Il comando seguente crea un sorgente Java:

```
create java source named "Hello" as
public class Hello {
    public static String hello() {
        return "Hello World";    } );
```

## CREATE LIBRARY

**VEDERE ANCHE** CREATE FUNCTION, CREATE PACKAGE BODY, CREATE PROCEDURE, Capitolo 29.

### SINTASSI

```
CREATE [OR REPLACE] LIBRARY [schema .] nomeLib
{ IS | AS } 'specificafайл' [AGENT 'dblink_agente'];
```

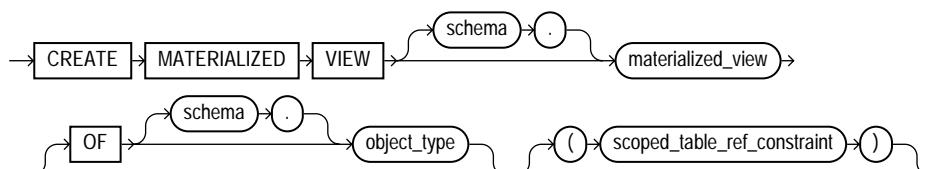
**DESCRIZIONE** CREATE LIBRARY crea un oggetto di libreria, consentendo di fare riferimento a una libreria condivisa del sistema operativo, dalla quale SQL e PL/SQL possano richiamare funzioni e procedure 3GL esterne. Per utilizzare le procedure e le funzioni memorizzate nella libreria, è necessario disporre del privilegio EXECUTE sulla libreria stessa. Si specifichi la clausola AGENT se si desidera che le procedure esterne vengano eseguite da un database link diverso dal server. Oracle si servirà del database link specificato da *dblink\_agente* per eseguire la procedure esterna. Se si tralascia questa clausola, l'agente predefinito sul server (extproc) eseguirà le procedure esterne.

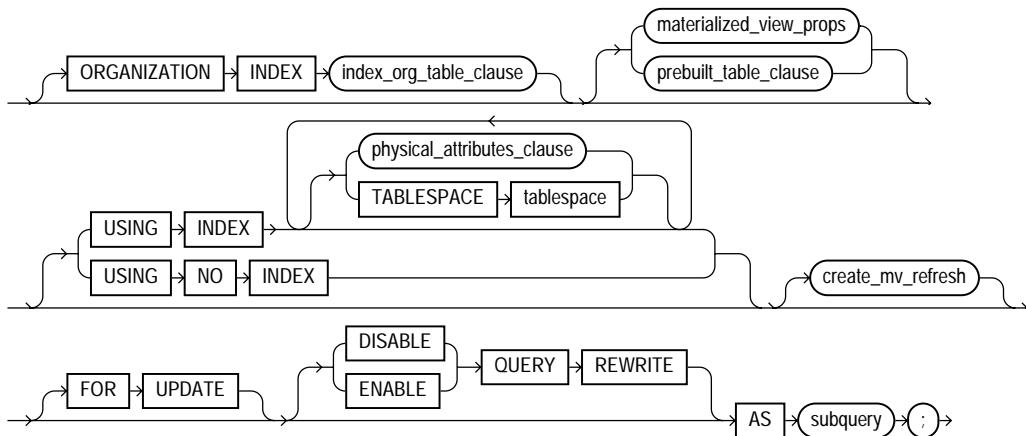
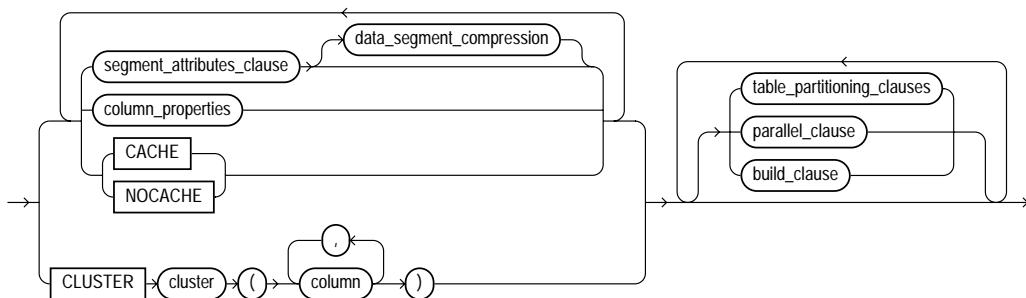
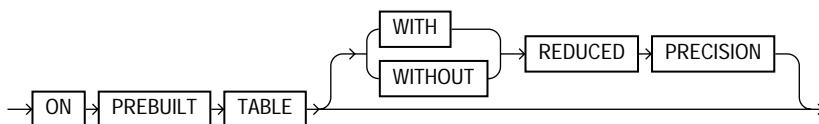
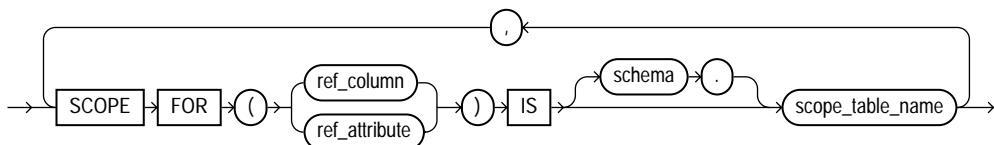
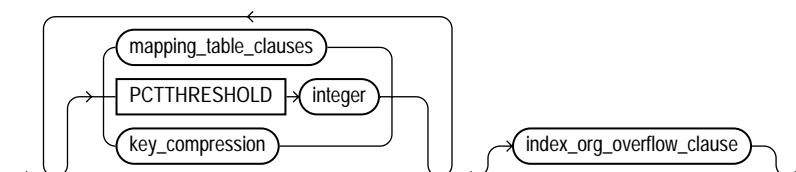
## CREATE MATERIALIZED VIEW

**VEDERE ANCHE** ALTER MATERIALIZED VIEW, CREATE MATERIALIZED VIEW LOG, DROP MATERIALIZED VIEW LOG, STORAGE Capitolo 23.

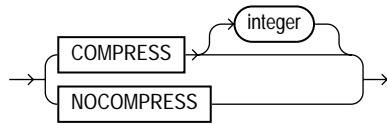
### SINTASSI

```
create_materialized_view
```



**materialized\_view\_props****prebuilt\_table\_clause****scoped\_table\_ref\_constraint****index\_org\_table\_clause**

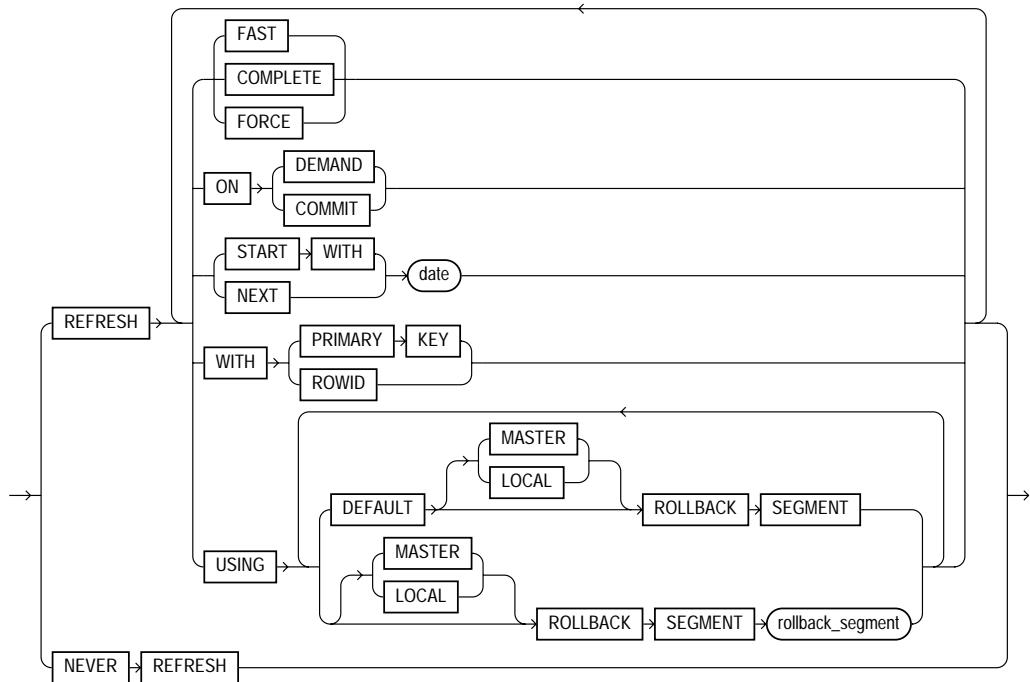
## key\_compression



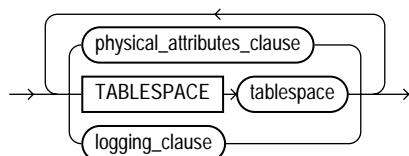
## index\_org\_overflow\_clause

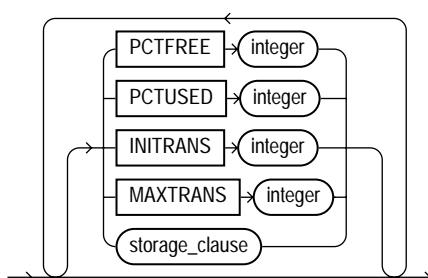
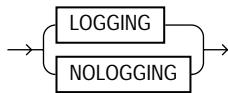
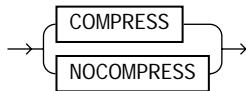
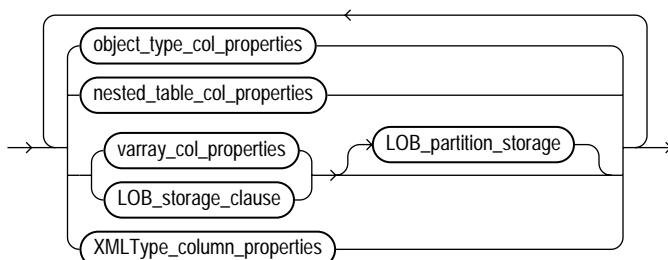
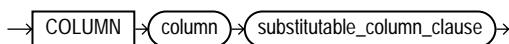
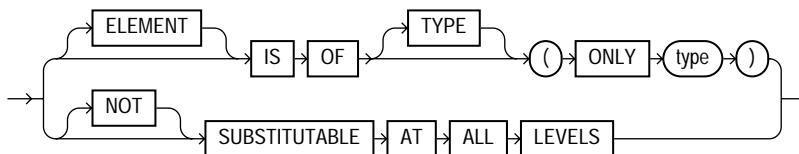
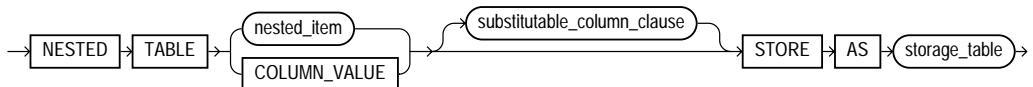


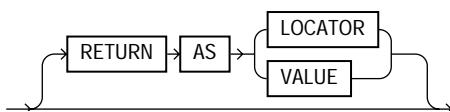
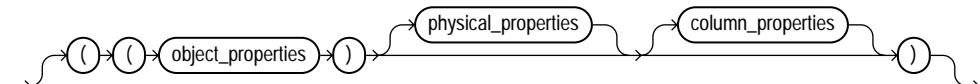
## create\_mv\_refresh



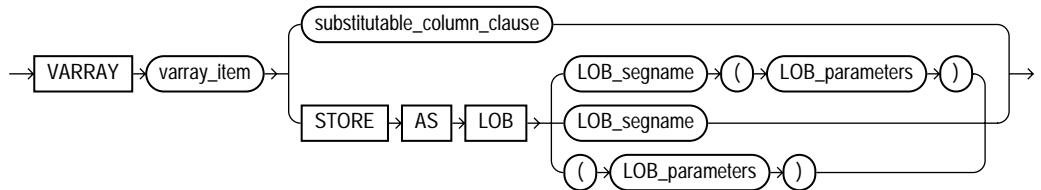
### segment\_attributes\_clause



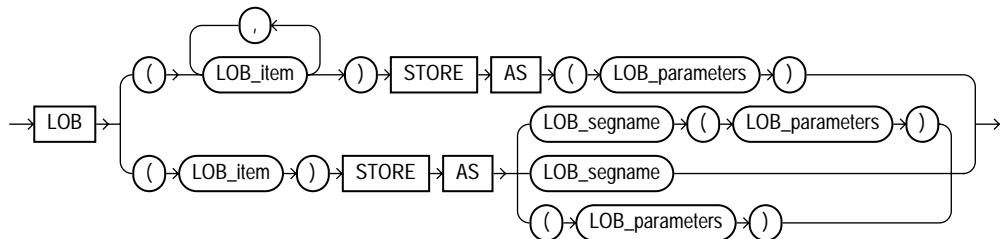
**physical\_attributes\_clause****logging\_clause****data\_segment\_compression****column\_properties****object\_type\_col\_properties****substitutable\_column\_clause****nested\_table\_col\_properties**



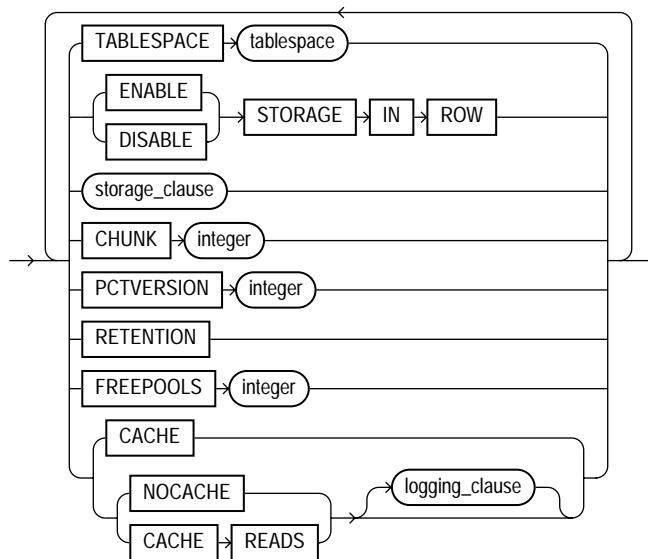
### `varray_col_properties`

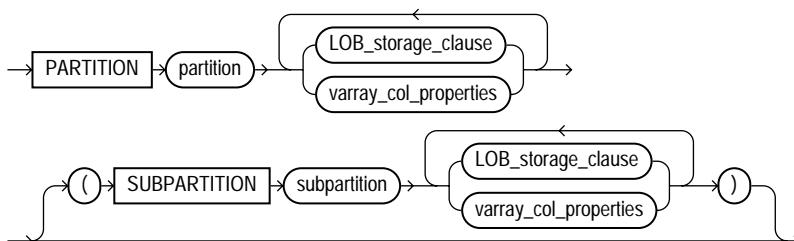
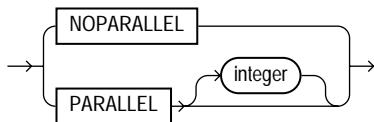
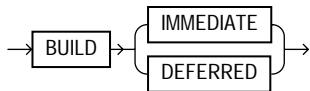


### `LOB_storage_clause`



### `LOB_parameters`



**LOB\_partition\_storage****parallel\_clause****build\_clause**

**DESCRIZIONE** La parola chiave `SNAPSHOT` è supportata al posto di `MATERIALIZED VIEW` per garantire la compatibilità con le versioni precedenti.

`CREATE MATERIALIZED VIEW` crea una vista materializzata, ovvero una tabella che contiene i risultati di una query, solitamente su una o più tabelle, chiamate tabelle master. Le tabelle master possono trovarsi in un database locale oppure in un database remoto. Si possono aggiornare i dati a intervalli regolari con la clausola `REFRESH`.

Per abilitare una vista materializzata alla riscrittura delle query, è necessario il privilegio di sistema `QUERY REWRITE`. Se la vista materializzata si basa su oggetti contenuti in altri schemi, si deve disporre del privilegio `GLOBAL QUERY REWRITE`.

Le viste materializzate non possono contenere colonne di tipo `LONG` o riferimenti a qualsiasi oggetto posseduto da `SYS`.

Un aggiornamento rapido utilizza il log delle viste materializzate associato alla tabella(e) master per aggiornare la vista materializzata. Un aggiornamento completo esegue di nuovo la query. Un aggiornamento forzato consente a Oracle di scegliere tra un aggiornamento rapido e uno completo. Per prima cosa Oracle aggiorna la vista materializzata alla *data* indicata tramite `START WITH`. Se si fornisce una *data* `NEXT`, Oracle aggiorna la vista materializzata a intervalli determinati dalla differenza tra le date `START WITH` e `NEXT`.

Una semplice vista materializzata seleziona i dati da un'unica tabella master utilizzando una query semplice. Una vista materializzata complessa seleziona i dati utilizzando una subquery `GROUP BY` o `CONNECT BY`, un join oppure operazioni di impostazione nella query. Oracle può eseguire un aggiornamento rapido solo sulle viste materializzate semplici che hanno log delle viste materializzate.

A causa degli oggetti locali creati dalle viste materializzate, il nome di una vista materializzata non dovrebbe superare i 19 caratteri di lunghezza. Per creare una vista materializzata nel proprio

schema, è necessario disporre del privilegio di sistema CREATE SNAPSHOT o CREATE MATERIALIZED VIEW. Per creare una vista materializzata nello schema di un altro utente, si deve avere il privilegio di sistema CREATE ANY SNAPSHOT o CREATE ANY MATERIALIZED VIEW.

### ESEMPIO

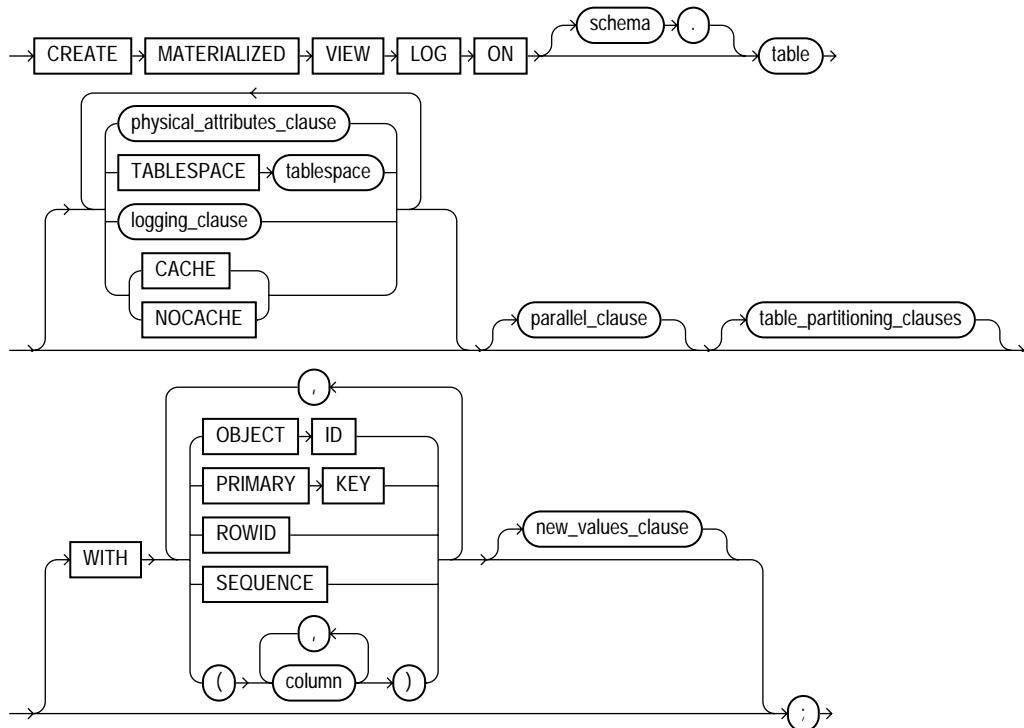
```
create materialized view BIBLIOTECA_LOCALE
storage (initial 100K next 100K pctincrease 0)
tablespace UTENTI
refresh force
start with SysDate next SysDate+7
with primary key
as
select * from BIBLIOTECA@REMOTE_CONNECT;
```

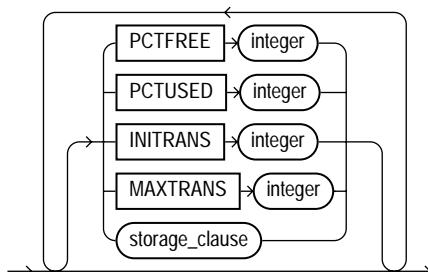
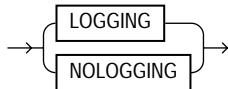
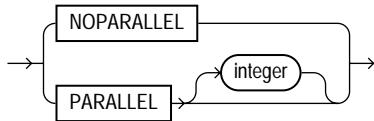
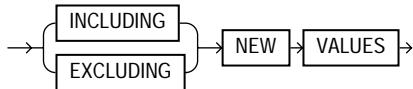
## CREATE MATERIALIZED VIEW LOG

**VEDERE ANCHE** ALTER MATERIALIZED VIEW LOG, CREATE MATERIALIZED VIEW, DROP MATERIALIZED VIEW LOG, STORAGE Capitolo 23.

### SINTASSI

`create_materialized_vw_log`



**physical\_attributes\_clause****logging\_clause****parallel\_clause****new\_values\_clause**

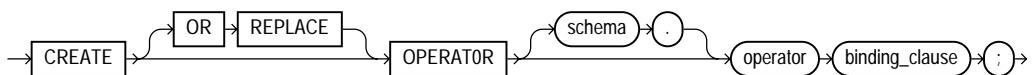
**DESCRIZIONE** La parola chiave `SNAPSHOT` è supportata al posto di `MATERIALIZED VIEW` per garantire la compatibilità con le versioni precedenti.

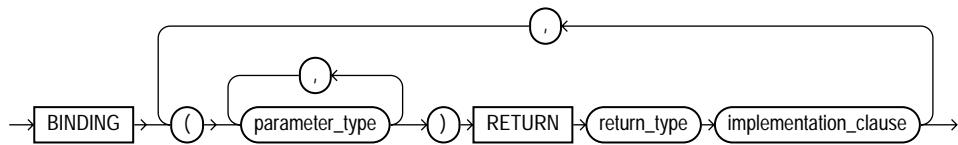
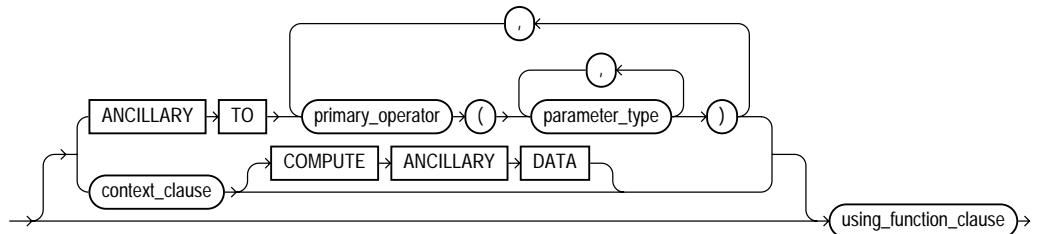
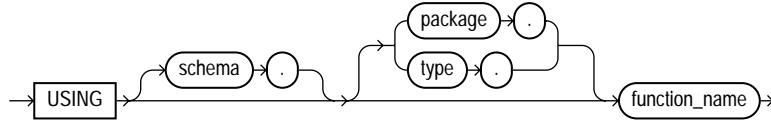
`CREATE MATERIALIZED VIEW LOG` crea una tabella associata alla tabella master di una vista materializzata che tiene traccia delle modifiche apportate ai dati della tabella master. Oracle usa il log delle viste materializzate per eseguire un aggiornamento rapido delle viste materializzate di una tabella master. Le opzioni di memorizzazione specificano la memorizzazione della tabella. Oracle registra le modifiche apportate al database solo se esiste una vista materializzata semplice che si basa sulla tabella master.

Per una determinata tabella master si può avere un solo log, che viene memorizzato nello stesso schema della tabella master. Per creare un log delle viste materializzate sulla propria tabella master, si deve possedere il privilegio di sistema `CREATE TABLE`. Se la tabella master si trova nello schema di un altro utente, si deve disporre dei privilegi di sistemi `CREATE ANY TABLE` e `COMMENT ANY TABLE` così come del privilegio `SELECT` sulla tabella master.

## CREATE OPERATOR

### SINTASSI

**create\_operator**


**binding\_clause****implementation\_clause****context\_clause****using\_function\_clause**

**DESCRIZIONE** CREATE OPERATOR crea un nuovo operatore e ne definisce i binding. Si può fare riferimento agli operatori nei tipi di indici e nelle istruzioni SQL. Gli operatori a loro volta fanno riferimento alle funzioni, ai package, ai tipi e ad altri oggetti definiti dall'utente.

Per creare un operatore, è necessario disporre del privilegio EXECUTE sulle funzioni e sugli operatori cui l'operatore fa riferimento, nonché del privilegio di sistema CREATE OPERATOR. Se l'operatore viene creato sullo schema di un altro utente, è necessario il privilegio di sistema CREATE ANY OPERATOR.

## CREATE OUTLINE

**VEDERE ANCHE** Capitolo 38.

### SINTASSI

```
CREATE [OR REPLACE] [ PUBLIC | PRIVATE ] OUTLINE [struttura]
[FROM [ PUBLIC | PRIVATE ] struttura_sorgente]
[FOR CATEGORY categoria]
[ON istruzione];
```

**DESCRIZIONE** CREATE OUTLINE crea una struttura memorizzata, ovvero un gruppo di suggerimenti utilizzati per creare un piano esecutivo della query associata. Le esecuzioni successive della query applicheranno lo stesso gruppo di suggerimenti. Le strutture memorizzate possono essere raggruppate in categorie.

Per creare un operatore, si deve possedere il privilegio di sistema CREATE ANY OUTLINE.

## ESEMPIO

```
create outline TEST
  for category DEVELOPMENT
    on select NomeAutore, COUNT(NomeAutore) from BIBLIOTECA_AUTORE group by NomeAutore;
```

## CREATE PACKAGE

**VEDERE ANCHE** ALTER PACKAGE; CREATE FUNCTION; CREATE PACKAGE BODY; CREATE PROCEDURE; CURSOR; DROP PACKAGE; ECCEZIONE; RECORD; TABELLA; VARIABILI, DICHIARAZIONE, Capitolo 29.

### SINTASSI

```
CREATE [OR REPLACE] PACKAGE [schema .] package
[AUTHID { CURRENT_USER | DEFINER }]
{IS | AS} spec_package_p1/sql;
```

**DESCRIZIONE** CREATE PACKAGE configura la specifica per un package PL/SQL, ovvero un gruppo di procedure, funzioni, eccezioni, variabili, costanti e cursori pubblici. L'aggiunta di OR REPLACE comporta la sostituzione della specifica del package se ne esiste già una; questa operazione invalida il package e avvia la ricompilazione del corpo di quest'ultimo e di qualsiasi oggetto dipendente dalla specifica del package.

L'inserimento nel package delle procedure e funzioni consente loro di condividere dati attraverso variabili, costanti e cursori. In questo modo, si ha la possibilità di accedere all'intera collezione in una volta sola come parte di un ruolo. Oracle accede a tutti gli elementi di un package in modo più efficiente di quanto farebbe se questi fossero separati. Se si modifica il corpo del package, che Oracle memorizza separatamente dalla specifica del package, non sarà necessario ricompilare nulla di ciò che utilizza il package.

Per poter creare un package, si deve disporre del privilegio di sistema CREATE PROCEDURE; per creare un package nell'account di un altro utente, occorre possedere il privilegio di sistema CREATE ANY PROCEDURE.

La clausola AUTHID consente di specificare se il codice del package viene eseguito con i privilegi del proprietario del package oppure con quelli dell'utente che esegue il codice del package.

## CREATE PACKAGE BODY

**VEDERE ANCHE** ALTER PACKAGE; CREATE FUNCTION; CREATE LIBRARY; CREATE PACKAGE; CREATE PROCEDURE; CURSOR; DROP PACKAGE; ECCEZIONE; RECORD; TABELLA; VARIABILI, DICHIARAZIONE; Capitoli 27 e 29.

### SINTASSI

```
CREATE [OR REPLACE] PACKAGE BODY [schema .] package
{IS | AS} corpo_package_p1/sql;
```

**DESCRIZIONE** CREATE PACKAGE BODY costruisce il corpo di un package precedentemente specificato, che è stato creato con CREATE PACKAGE. Quando si aggiunge OR REPLACE, viene sostituito il corpo del package, se già esiste. Per creare il corpo di un package, è necessario

disporre del privilegio di sistema CREATE PROCEDURE; per creare il corpo di un package nell'account di un altro utente, occorre possedere il privilegio di sistema CREATE ANY PROCEDURE.

## CREATE PFILE

**VEDERE ANCHE** CREATE SPFILE, init.ora.

### SINTASSI

```
CREATE PFILE [= 'nome_filep'] FROM SPFILE [= 'nome_spfile'];
```

**DESCRIZIONE** CREATE PFILE esporta un file binario dei parametri di server in un file di testo dei parametri di inizializzazione (init.ora). La creazione di un file di testo dei parametri è una soluzione conveniente per ottenere un elenco delle impostazioni dei parametri correnti utilizzate dal database, e consente di modificare facilmente il file in un editore di testo e di convertirlo nuovamente in un file dei parametri di server con l'istruzione CREATE SPFILE. Si possono specificare entrambi i nomi: il nome del file binario di origine dei parametri di server e quello del file di testo di destinazione dei parametri.

### ESEMPIO

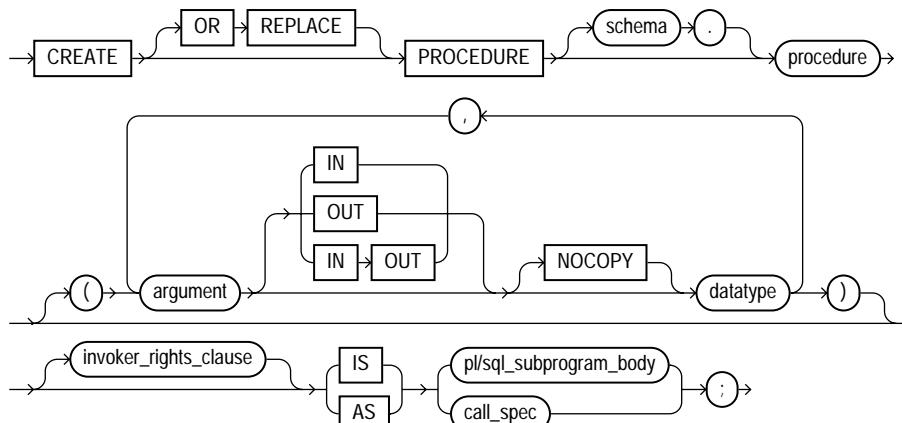
```
create pfile from spfile;
```

## CREATE PROCEDURE

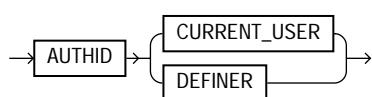
**VEDERE ANCHE** ALTER PROCEDURE; BLOCCO, STRUTTURA; CREATE LIBRARY; CREATE FUNCTION; CREATE PACKAGE; DATI, TIPI; DROP PROCEDURE, Capitoli 27 e 29.

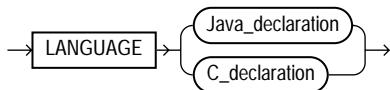
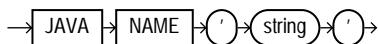
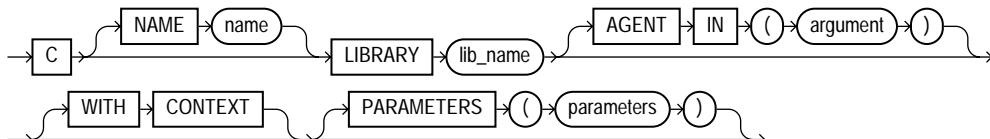
### SINTASSI

```
create_procedure
```



**invoker\_rights\_clause**



**call\_spec****Java\_declaration****C\_declaration**

**DESCRIZIONE** CREATE PROCEDURE crea la specifica e il corpo di una procedura. Una procedura può avere parametri, ovvero argomenti di un certo tipo di dati cui è stato assegnato un nome. Il blocco PL/SQL definisce il comportamento della procedura come una serie di dichiarazioni, istruzioni di programma PL/SQL ed eccezioni.

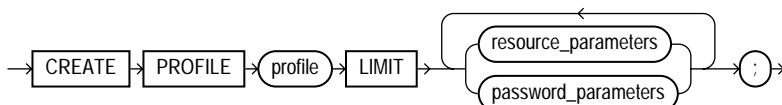
Il qualificatore IN significa che è necessario specificare un valore per l'argomento quando si chiama la procedura. Il qualificatore OUT significa che la procedura restituisce un valore al chiamante attraverso questo argomento. Il qualificatore IN OUT combina sia il significato di IN che quello di OUT; si specifica un valore e la procedura lo sostituisce con un altro. Se non si specifica alcun qualificatore, l'argomento predefinito è IN. La differenza tra una funzione e una procedura è che la funzione restituisce un valore all'ambiente chiamante oltre a qualsiasi argomento di tipo OUT.

Il blocco PL/SQL può fare riferimento ai programmi contenuti in una libreria C esterna oppure a una specifica di chiamata Java. La clausola *invoker\_rights* (AUTHID) consente di specificare se la procedura viene eseguita con i privilegi del proprietario della funzione o con quelli dell'utente che chiama la funzione.

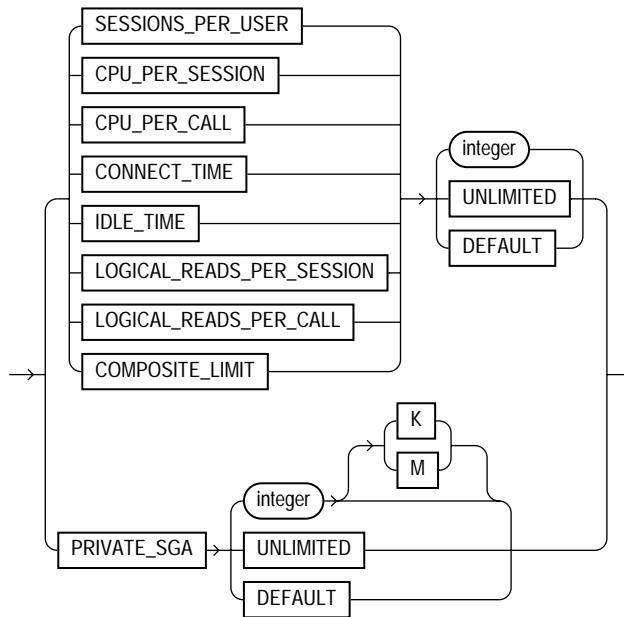
## CREATE PROFILE

**VEDERE ANCHE** ALTER PROFILE, ALTER RESOURCE COST, ALTER SYSTEM, ALTER USER, CREATE USER, DROP PROFILE, Capitolo 19.

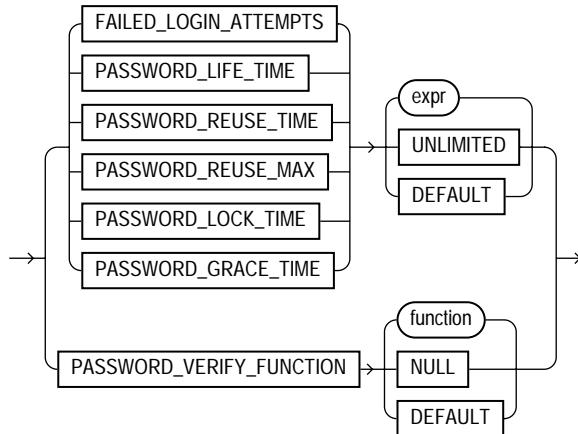
### SINTASSI

**create\_profile**

### resource\_parameters



### password\_parameters



**DESCRIZIONE** `CREATE PROFILE` crea un insieme di limiti relativi all'uso delle risorse del database. Quando si associa il profilo a un utente tramite `CREATE` o `ALTER USER`, è possibile controllare quello che l'utente fa grazie a questi limiti,. Per utilizzare `CREATE PROFILE`, è necessario abilitare i limiti delle risorse attraverso il parametro di inizializzazione `RESOURCE_LIMIT` oppure con il comando `ALTER SYSTEM`.

`SESSIONS_PER_USER` limita il numero di sessioni SQL contemporanee *interne* per l'utente. `CPU_PER_SESSION` limita il tempo di CPU in centesimi di secondo. `CPU_PER_CALL`

limita il tempo di CPU in centesimi di secondo per analisi, esecuzioni o chiamate fetch. CONNECT\_TIME limita il tempo rimanente di una sessione in minuti. IDLE\_TIME scollega un utente trascorsi i minuti specificati; ciò non si applica durante l'esecuzione di una query. LOGICAL\_READS\_PER\_SESSION limita il numero di blocchi letti per sessione; LOGICAL\_READS\_PER\_CALL fa la stessa cosa per l'analisi, l'esecuzione o le chiamate fetch. PRIVATE\_SGA limita la quantità di spazio che è possibile allocare come privato nella SG; le opzioni K e M si applicano solo a questo limite. COMPOSITE\_LIMIT limita il costo totale delle risorse per sessione, espresso in unità di servizio in base a una somma pesata dei costi della CPU, del tempo di connessione, delle letture logiche e delle risorse SGA private.

UNLIMITED significa che non esiste un limite per una particolare risorsa. DEFAULT seleziona il limite dal profilo predefinito, che può essere modificato tramite il comando ALTER PROFILE.

Se un utente supera un limite, Oracle sospende l'esecuzione ed esegue il rollback della transazione, quindi termina la sessione. Per creare un profilo, è necessario disporre del privilegio di sistema CREATE PROFILE. Per associare un profilo a un utente, si utilizza il comando ALTER USER.

## CREATE ROLE

**VEDERE ANCHE** ALTER ROLE, ALTER USER, CREATE USER, DROP ROLE, GRANT, REVOKE, ROLE, SET ROLE, Capitolo 19.

### SINTASSI

```
CREATE ROLE ruolo
[ NOT IDENTIFIED
| IDENTIFIED {BY password |
  USING [schema .] package | EXTERNALLY | GLOBALLY}];
```

**DESCRIZIONE** Con CREATE ROLE è possibile creare un ruolo con un certo nome o un insieme di privilegi. Quando si concede un ruolo a un utente, in realtà gli si concedono tutti i privilegi di quel ruolo. Per prima cosa, è necessario creare il ruolo con CREATE ROLE, poi concedere i privilegi al ruolo con il comando GRANT. Quando un utente desidera accedere a qualche operazione consentita dal ruolo che possiede, abilita il ruolo con SET ROLE. In alternativa, si può impostare il ruolo come il ruolo DEFAULT dell'utente tramite il comando ALTER USER o CREATE USER.

Se si protegge il ruolo tramite una password, l'utente che intende disporre dei privilegi deve fornire la password con il comando SET ROLE per abilitare il ruolo. La clausola USING *package* consente di creare il ruolo di un'applicazione, ovvero un ruolo che può essere abilitato soltanto da applicazioni che usano il package autorizzato. Se non si specifica *schema*, Oracle assume che il package si trovi nello schema di proprietà dell'utente.

## CREATE ROLLBACK SEGMENT

**VEDERE ANCHE** ALTER ROLLBACK SEGMENT, CREATE DATABASE, CREATE TABLESPACE, DROP ROLLBACK SEGMENT, STORAGE, Capitolo 40.

### SINTASSI

```
CREATE [PUBLIC] ROLLBACK SEGMENT segmento_rollback
[ { TABLESPACE tablespace | storage_clause }
[ TABLESPACE tablespace | storage_clause ] ...];
```

**DESCRIZIONE** *segmento\_rollback* è il nome assegnato a questo segmento di rollback. *tablespace* è il nome della tablespace cui è assegnato questo segmento di rollback. Una tablespace può avere più segmenti di rollback.

*storage\_clause* contiene le clausole secondarie descritte alla voce STORAGE. Se si usa PUBLIC, questo segmento di rollback può essere utilizzato da qualsiasi istanza lo richieda; in caso contrario, è disponibile solo per le istanze che lo citano esplicitamente nei rispettivi file dei parametri di inizializzazione. Per i dettagli sull'impiego e la gestione dei segmenti di rollback, si consulti il Capitolo 40.

## CREATE SCHEMA

**VEDERE ANCHE** CREATE TABLE, CREATE VIEW, GRANT.

### SINTASSI

```
CREATE SCHEMA AUTHORIZATION schema
{ istruzione_creazione_tabella | istruzione_creazione_vista | istruzione_concessione }
[istruzione_creazione_tabella | istruzione_creazione_vista | istruzione_concessione]...;
```

**DESCRIZIONE** Il comando CREATE SCHEMA crea una collection di tabelle, viste e concessioni di privilegi come un'unica transazione. Il nome dello schema è identico al proprio nome utente di Oracle. I comandi CREATE TABLE, CREATE VIEW e GRANT sono i comandi standard e l'ordine con cui appaiono non è importante, anche se esistono dipendenze interne.

## CREATE SEQUENCE

**VEDERE ANCHE** ALTER SEQUENCE, AUDIT, DROP SEQUENCE, GRANT, REVOKE, NEXTVAL e CURRVAL alla voce PSEUDOCOLONNE, Capitolo 20.

### SINTASSI

```
CREATE SEQUENCE [schema .] sequenza
[ { { INCREMENT BY | START WITH } intero
| { MAXVALUE intero | NOMAXVALUE }
| { MINVALUE intero | NOMINVALUE }
| { CYCLE | NOCYCLE }
| { CACHE intero | NOCACHE }
| { ORDER | NOORDER }
}
[ { INCREMENT BY | START WITH } intero
| { MAXVALUE intero | NOMAXVALUE }
| { MINVALUE intero | NOMINVALUE }
| { CYCLE | NOCYCLE }
| { CACHE intero | NOCACHE }
| { ORDER | NOORDER } ]... ];
```

**DESCRIZIONE** *sequenza* è il nome assegnato alla sequenza. Il valore predefinito per INCREMENT BY è 1. Un numero positivo incrementa il numero della sequenza di un intervallo uguale all'intero. Un numero negativo decremente la sequenza (ossia ne riduce il valore) nello stesso modo. START WITH è il numero con cui la sequenza ha inizio. Il valore predefinito per START WITH è MAXVALUE per le sequenze in ordine decrescente e MINVALUE per quelle in ordine crescente; per sostituire questo valore predefinito, si utilizzi START WITH.

**MINVALUE** è il numero più piccolo che la sequenza può generare. Il valore predefinito è 1 per le sequenze in ordine crescente. **MAXVALUE** è il numero più grande che la sequenza può generare. Per le sequenze in ordine decrescente il valore predefinito è -1. Per consentire alle sequenze di progredire senza limiti, è sufficiente specificare **MINVALUE** per le sequenze in ordine crescente e **MAXVALUE** per quelle in ordine decrescente. Per interrompere la creazione dei numeri di sequenza e indurre un errore nel caso si cerchi di ottenere un nuovo numero, occorre specificare un **MAXVALUE** su una sequenza in ordine crescente, o un **MINVALUE** su una sequenza in ordine decrescente, più **NOCYCLE**. Per riavviare entrambi i tipi di sequenze dal punto corrispondente al relativo **MAXVALUE** o **MINVALUE**, occorre specificare **CYCLE**.

**CACHE** consente di tenere in memoria un gruppo preallocato di numeri di sequenza. Il valore predefinito è 20. Il valore impostato dev'essere minore di **MAXVALUE** meno **MINVALUE**.

**ORDER** garantisce che i numeri di sequenza verranno assegnati alle istanze che li richiedono nell'ordine in cui vengono ricevute le richieste. Ciò è necessario solamente nelle applicazioni che utilizzano Real Application Clusters.

Per creare una sequenza nel proprio schema, è necessario disporre del privilegio di sistema **CREATE SEQUENCE**; per creare una sequenza nello schema di un altro utente occorre possedere il privilegio di sistema **CREATE ANY SEQUENCE**.

## CREATE SNAPSHOT

Vedere **CREATE MATERIALIZED VIEW**.

## CREATE SNAPSHOT LOG

Vedere **CREATE MATERIALIZED VIEW LOG**.

## CREATE SPFILE

**VEDERE ANCHE** **CREATE PFILE**, **init.ora**.

### SINTASSI

```
CREATE SPFILE [= 'nome_spfile'] FROM PFILE [= 'nome_pfile'];
```

**DESCRIZIONE** **CREATE SPFILE** crea un file dei parametri per server da un file dei parametri di inizializzazione testuale sul lato client. I file dei parametri per server sono file binari che esistono solo sul server e vengono chiamati dalle posizioni del client per avviare il database. I file dei parametri di server consentono di modificare in modo permanente i singoli parametri; le modifiche apportate ai parametri con i comandi **ALTER SYSTEM** vengono conservative durante le chiusure e gli avvii del database anche se nessun file dei parametri esterno viene aggiornato manualmente.

## CREATE SYNONYM

**VEDERE ANCHE** **CREATE DATABASE LINK**, **CREATE TABLE**, **CREATE VIEW**, Capitoli 19 e 22.

## SINTASSI

```
CREATE [PUBLIC] SYNONYM [schema .] sinonimo
FOR [schema .] oggetto [@ dblink];
```

**DESCRIZIONE** CREATE SYNONYM crea un sinonimo per una tabella, vista, sequenza, stored procedure, funzione, package, vista materializzata, oggetto di classe Java oppure sinonimo compresi quelli che risiedono su un server remoto. PUBLIC rende disponibile il sinonimo per tutti gli utenti, ma può essere creato solo da un DBA. Senza PUBLIC, gli utenti devono anteporre al sinonimo il proprio nome utente. *sinonimo* è il nome del sinonimo. *utente.tabella* rappresenta il nome della tabella, della vista o del sinonimo cui fa riferimento il sinonimo stesso. È possibile creare il sinonimo di un sinonimo. *@link\_database* è un database link a un database remoto. Il sinonimo si riferisce a una tabella nel database remoto come specificato dal database link.

## ESEMPIO

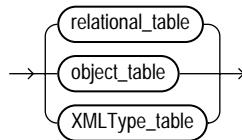
```
create synonym BIBLIOTECA_LOCALE for BIBLIOTECA@REMOTE_CONNECT;
```

## CREATE TABLE

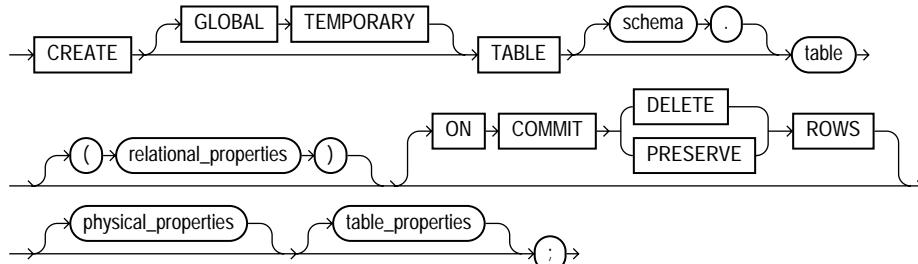
**VEDERE ANCHE** ALTER TABLE; CREATE CLUSTER; CREATE INDEX; CREATE TABLESPACE; CREATE TYPE; DATI, TIPI; DROP TABLE, VINCOLO DI INTEGRITÀ, NOMI DEGLI OGGETTI, STORAGE, Capitoli 3, 18, 20, 25, 30, 31 e 33.

## SINTASSI

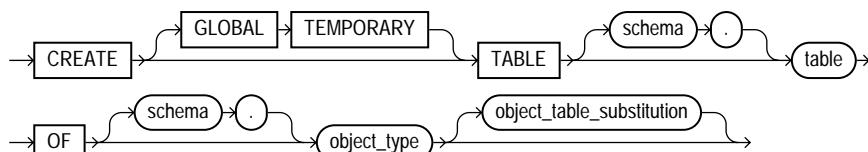
create\_table

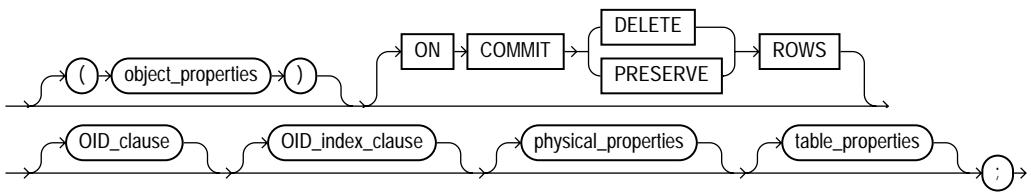
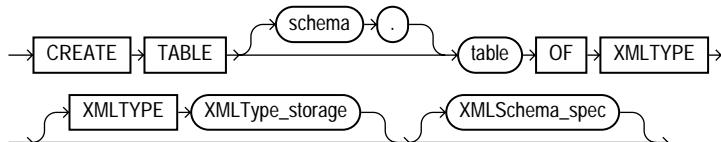
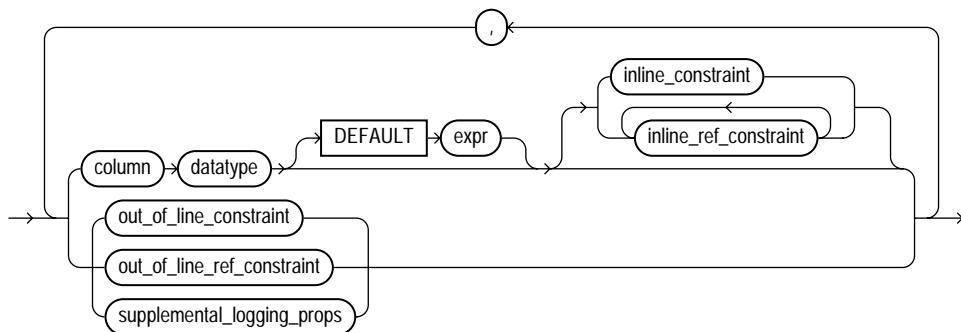
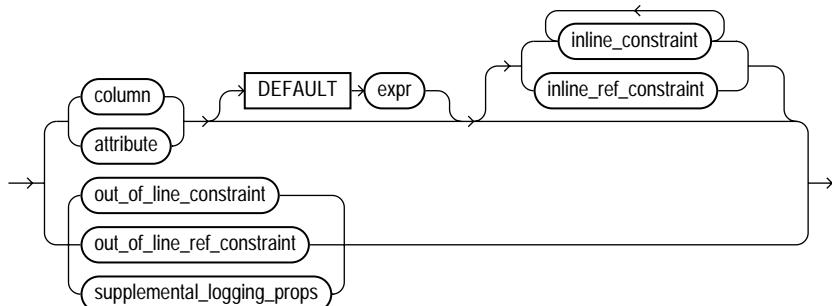


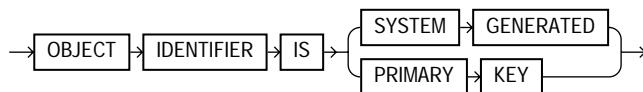
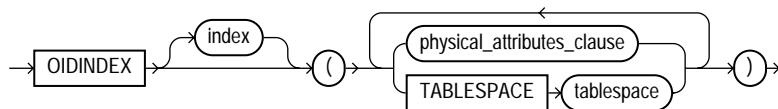
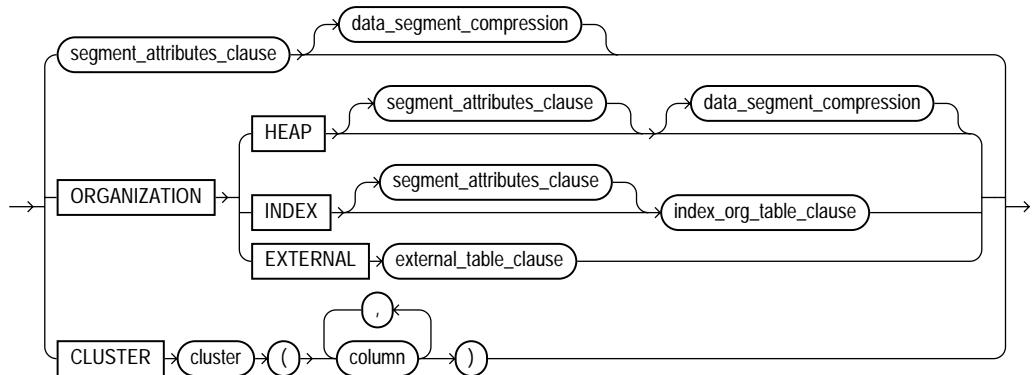
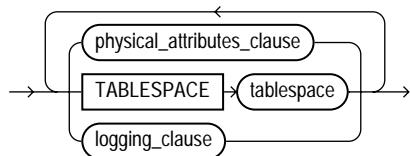
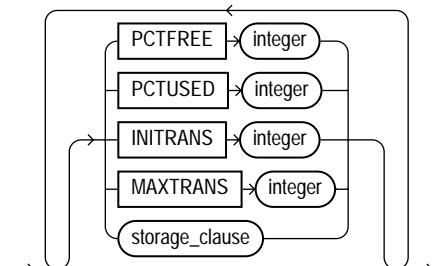
relational\_table

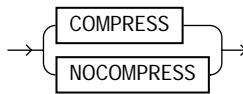
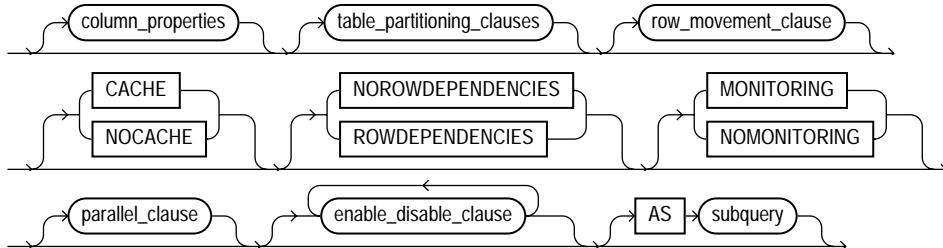
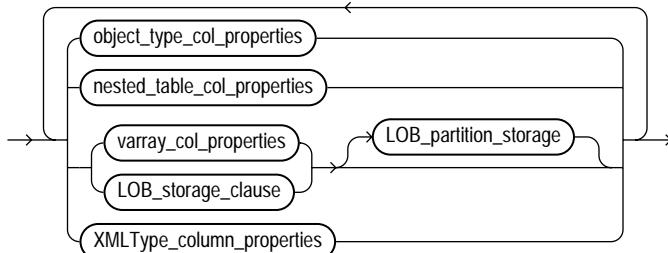
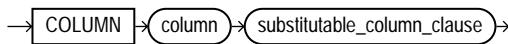
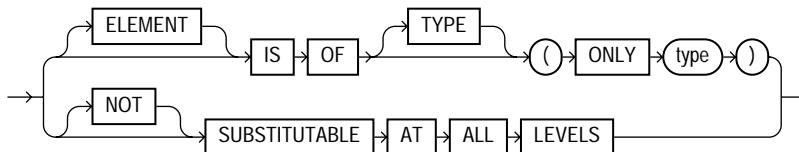
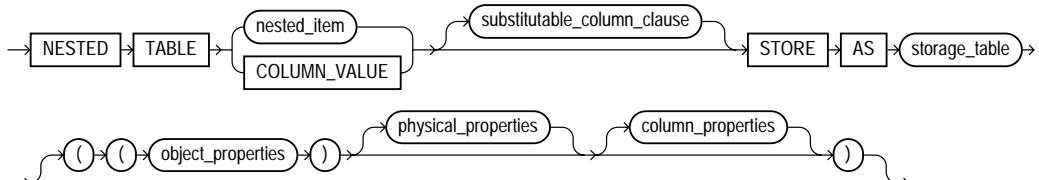


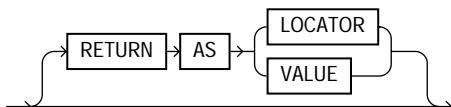
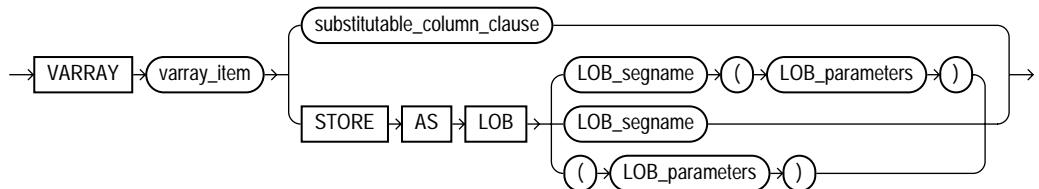
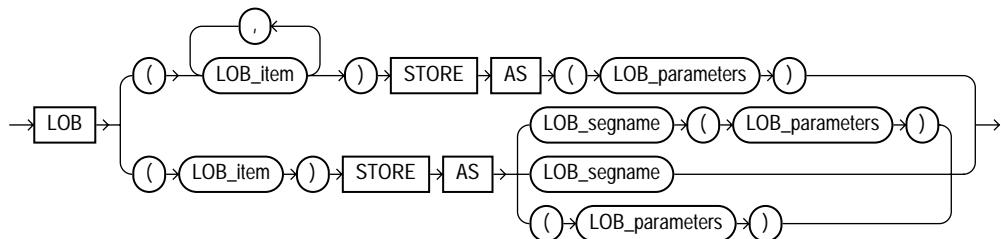
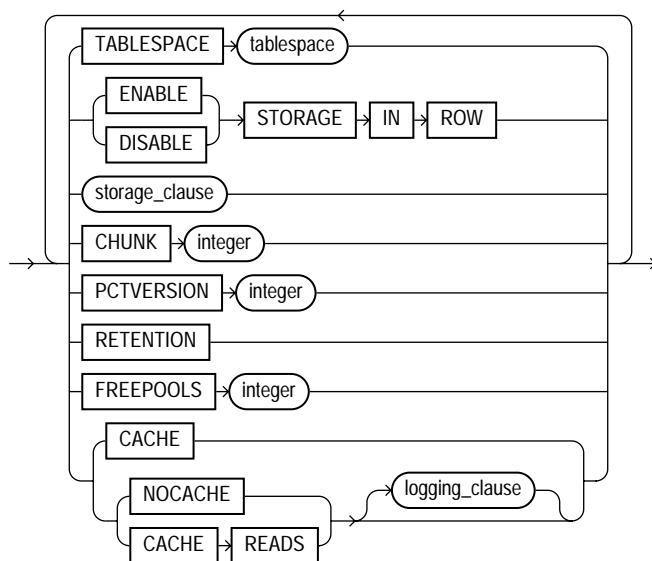
object\_table



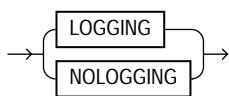
**XMLType\_table****relational\_properties****object\_table\_subs\_clause****object\_properties**

**oid\_clause****oid\_index\_clause****physical\_properties****segment\_attributes\_clause****physical\_attributes\_clause**

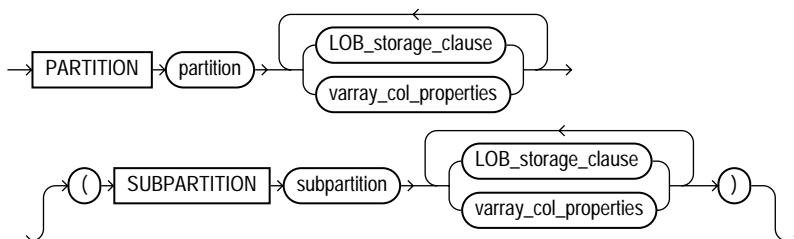
**data\_segment\_compression****table\_properties****column\_properties****object\_type\_col\_properties****substitutable\_column\_clause****nested\_table\_col\_properties**

**varray\_col\_properties****LOB\_storage\_clause****LOB\_parameters**

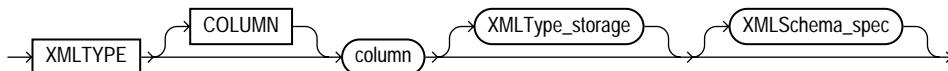
## logging\_clause



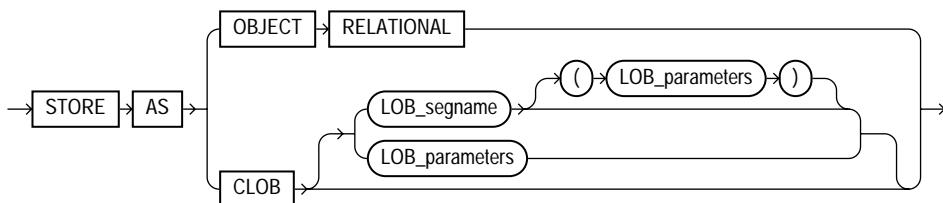
## LOB\_partition\_storage



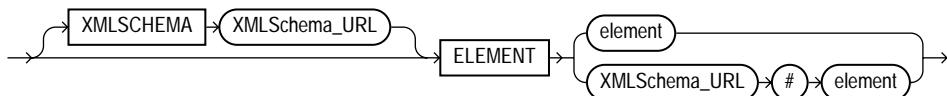
## **XMLType\_column\_properties**



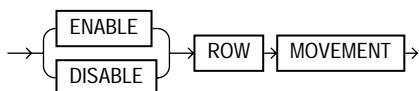
## XMLType storage



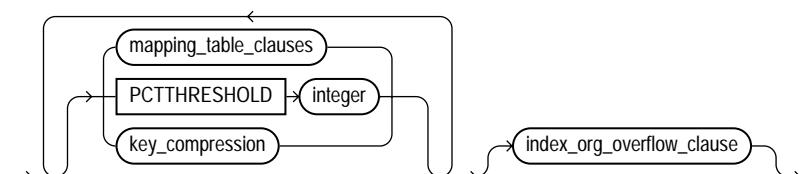
## XMLSchema\_spec

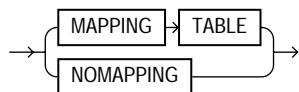
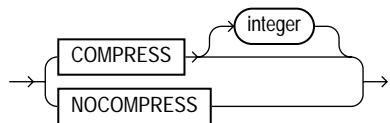
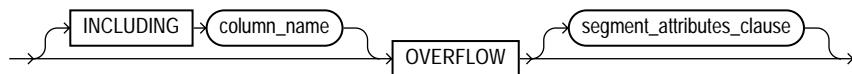
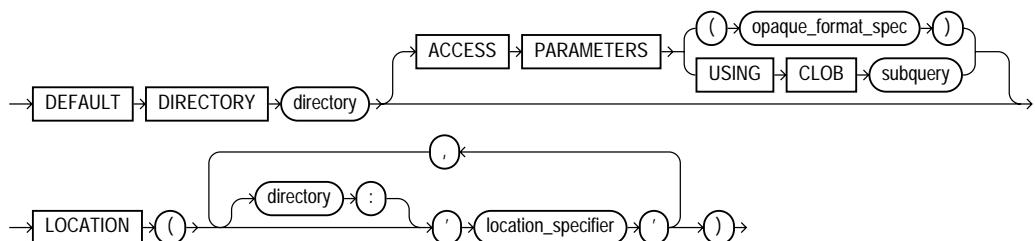
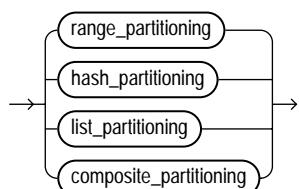


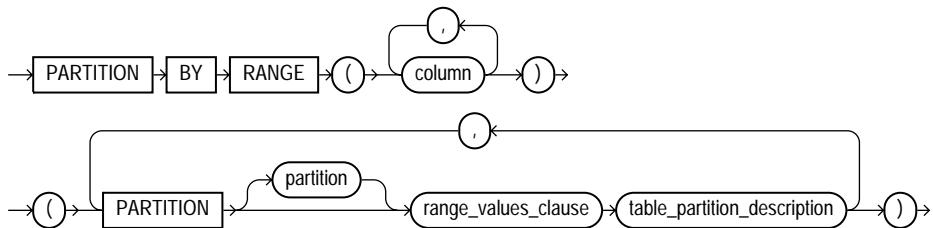
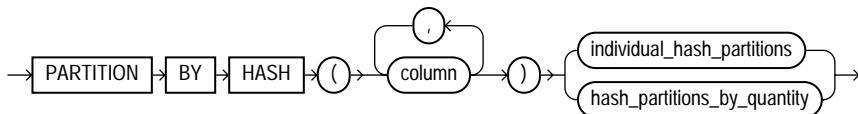
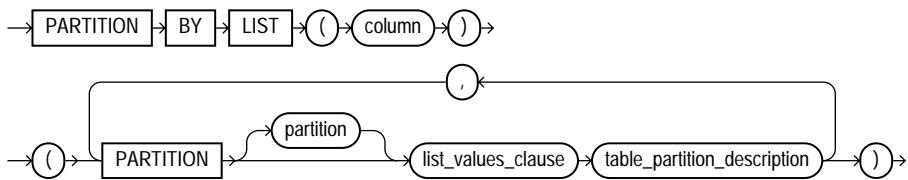
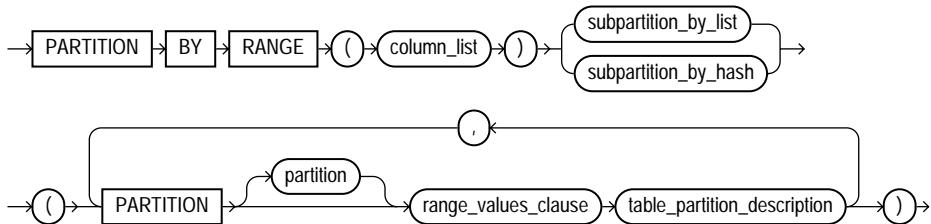
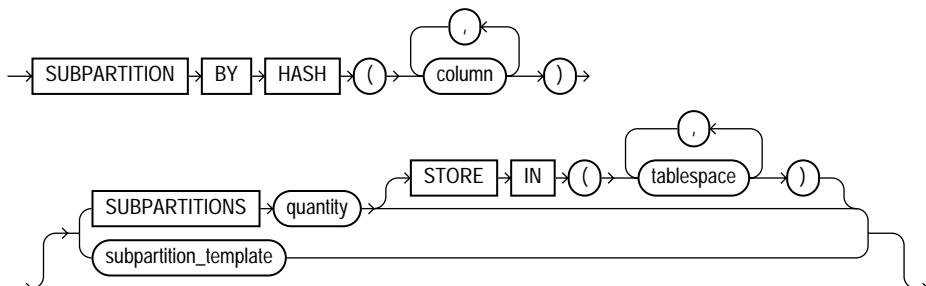
### **row\_movement\_clause**

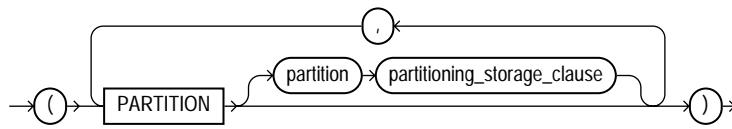
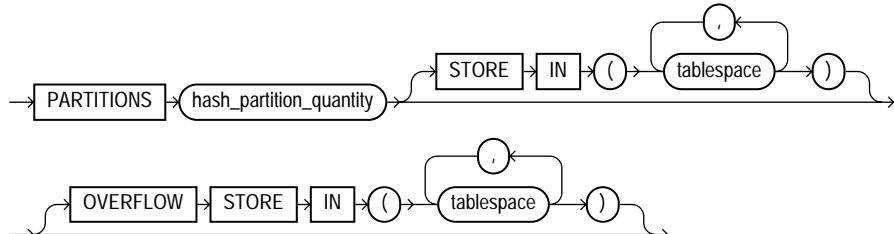
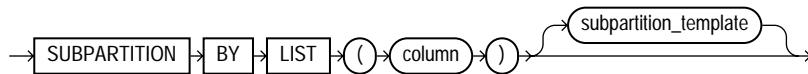
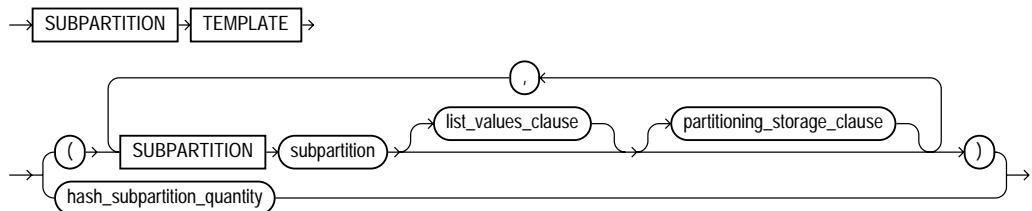
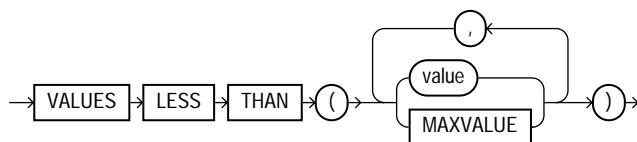
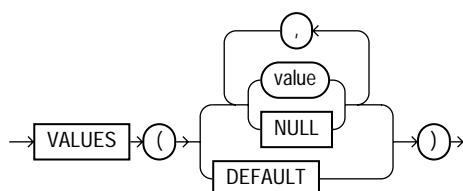


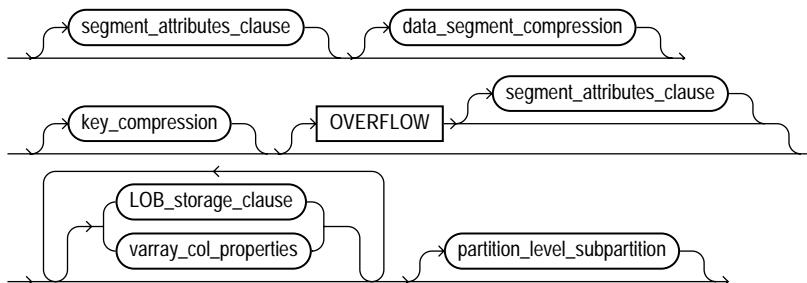
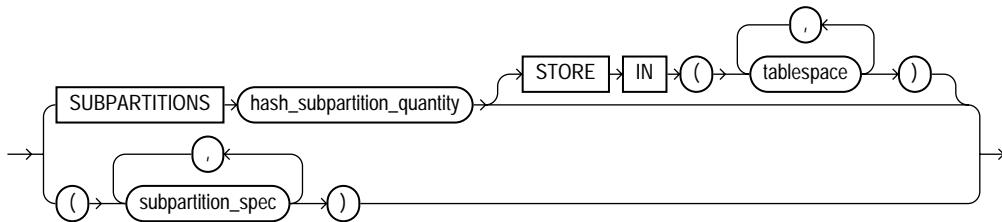
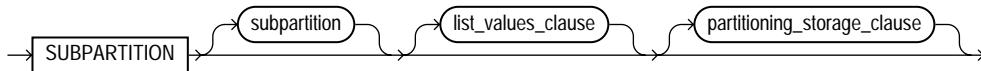
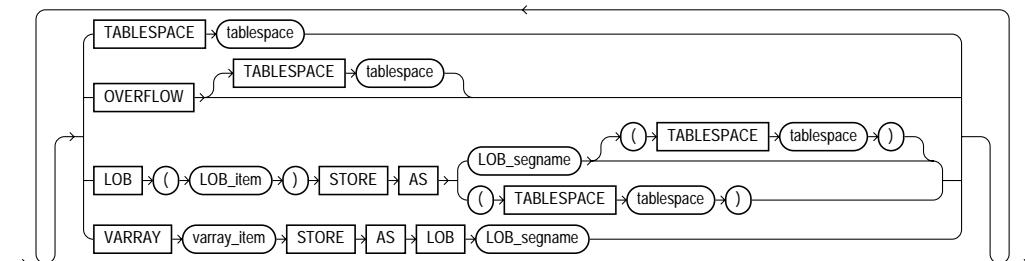
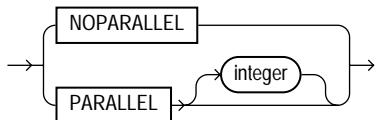
index org table clause

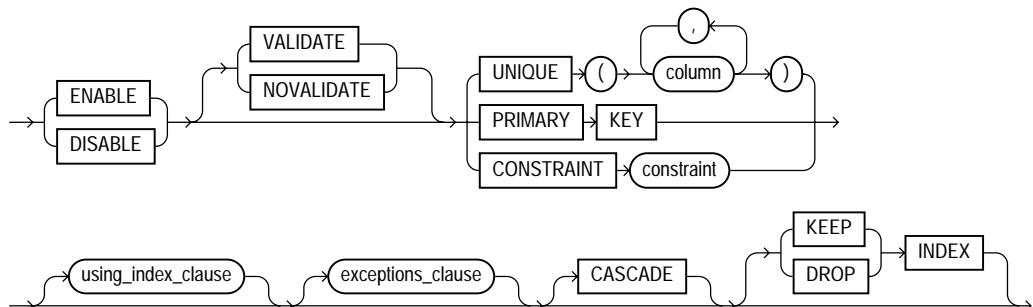
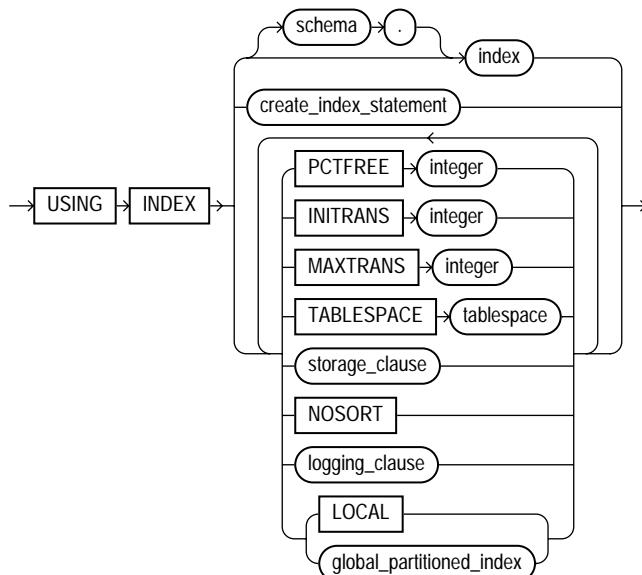
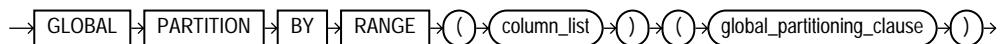
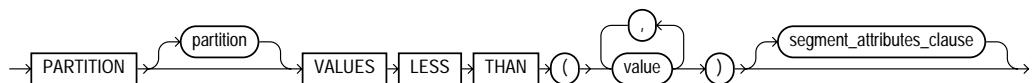


**mapping\_table\_clauses****key\_compression****index\_org\_overflow\_clause****supplemental\_logging\_props****external\_table\_clause****external\_data\_properties****table\_partitioning\_clauses**

**range\_partitioning****hash\_partitioning****list\_partitioning****composite\_partitioning****subpartition\_by\_hash**

**individual\_hash\_partitions****hash\_partitions\_by\_quantity****subpartition\_by\_list****subpartition\_template****range\_values\_clause****list\_values\_clause**

**table\_partition\_description****partition\_level\_subpartition****subpartition\_spec****partitioning\_storage\_clause****parallel\_clause**

**enable\_disable\_clause****using\_index\_clause****global\_partitioned\_index****global\_partitioning\_clause**

**DESCRIZIONE** *utente* rappresenta il proprietario della tabella. Se il proprietario della tabella viene omesso, per default *utente* viene impostato all'utente che ha eseguito il comando. *tabella* è il nome della tabella e segue le convenzioni di denominazione di Oracle.

*colonna* è il nome di una colonna, mentre *tipo di dati* è uno dei tipi di dati forniti da Oracle (vedere DATI, TIPI) oppure un tipo di dati definito dall'utente (si consulti CREATE TYPE).

**DEFAULT** specifica un valore da assegnare alla colonna se si inserisce una riga priva di valore per questa colonna. Il valore può essere di tipo letterale semplice o il risultato di un'espressione. L'espressione, tuttavia, non può includere un riferimento a una colonna, a Level o a RowNum.

Per un tipo di dati astratto già esistente, si può creare una tabella oggetto. L'esempio seguente crea una tabella oggetto di nome ANIMALE, che si basa sul tipo di dati ANIMALE\_TY:

```
create table ANIMALE of ANIMALE_TY;
```

Per i dettagli sull'uso delle tabelle oggetto, si rimanda al Capitolo 33.

*Vedere VINCOLO DI INTEGRITÀ* per una descrizione completa dei vincoli di tabella e di colonna.

**CLUSTER** include la tabella in questione nel cluster specificato. Le colonne della tabella elencate devono corrispondere, nell'ordine e nel tipo di dati, alle colonne del cluster. Non occorre che i nomi siano gli stessi delle corrispondenti colonne del cluster, anche se la corrispondenza dei nomi aiuterebbe l'utente a capire cosa succede.

**INITRANS** specifica il numero iniziale di transazioni che possono aggiornare contemporaneamente un blocco di dati (le selezioni non sono contate). La sequenza di valori per INITRANS va da 1 a 255. 1 è il valore predefinito. Ogni transazione occupa spazio nel blocco di dati (23 byte nella maggior parte dei sistemi) fino a che non risulta completata. Quando per un blocco si crea un numero di transazioni maggiore di INITRANS, lo spazio viene automaticamente allocato per queste transazioni, fino a un massimo pari a MAXTRANS.

**MAXTRANS** specifica il numero massimo di transazioni che possono aggiornare contemporaneamente un blocco di dati (le selezioni non sono contate). MAXTRANS va da 1 a 255. 255 è il valore predefinito. Ogni transazione occupa spazio (23 byte nella maggior parte dei sistemi) nel blocco di dati fino a che non risulta completata. Le code di transazioni in attesa di essere eseguite occupano sempre più spazio all'interno del blocco, anche se, solitamente, c'è più spazio libero di quanto necessario per tutte le transazioni contemporanee.

Ogni volta che Oracle inserisce una riga in una tabella, prima controlla per vedere quanto spazio è disponibile nel blocco corrente (la dimensione di un blocco dipende dal sistema operativo, si consulti *Oracle Installation and User's Guide* per il proprio sistema). Se la dimensione della riga lascia meno spazio della percentuale PCTFREE nel blocco, la riga viene inserita in un blocco appena allocato. Il valore predefinito è 10; 0 è il valore minimo.

**PCTUSED** ha un valore predefinito pari a 40, che rappresenta la percentuale minima di spazio disponibile in un blocco perché questo rappresenti un candidato per l'inserimento di nuove righe. Oracle tiene traccia di quanto spazio è stato liberato dalle cancellazioni in un blocco. Se questo valore è inferiore a PCTUSED, Oracle rende disponibile il blocco per nuovi inserimenti.

STORAGE contiene delle clausole secondarie descritte alla voce STORAGE. TABLESPACE è il nome della tablespace cui è assegnata questa tabella.

Le clausole **ENABLE** e **DISABLE** abilitano o disabilitano, rispettivamente, i vincoli.

**PARALLEL**, assieme a **DEGREE** e **INSTANCES**, specifica le caratteristiche parallele della tabella. **DEGREE** specifica il numero di server da utilizzare per le query; **INSTANCES** specifica il modo in cui la tabella deve essere suddivisa tra le istanze di Real Application Clusters per l'elaborazione in parallelo delle query. Un intero *n* indica che la tabella dev'essere suddivisa tra il numero specificato di istanze disponibili.

Le clausole **PARTITION** controllano come vengono partizionati i dati della tabella: per intervallo, per hash oppure con una composizione di più metodi di partizionamento. Per i dettagli sul partizionamento, si consulti il Capitolo 18.

Una tabella TEMPORARY contiene dei dati che sono visibili solo per la sessione che li ha creati. Se una tabella temporanea viene definita come GLOBAL la sua definizione può essere visibile a tutti gli utenti. Le tabelle temporanee possono anche non essere partizionate e non sono in grado di supportare molte caratteristiche standard delle tabelle (come i tipi di dati array variabile, i tipi di dati LOB e l'organizzazione di solo indice). Per esempio, il comando seguente crea una tabella le cui righe verranno cancellate alla fine della sessione corrente

```
create global temporary table PROGRAMMAZIONE_LAVORO (
    DataInizio DATE,
    DataFine   DATE,
    Tasso_Pagamento NUMBER)
on commit preserve rows;
```

La clausola AS crea le righe della nuova tabella attraverso le righe restituite dalla query. Le colonne e i tipi della query devono corrispondere a quelli definiti in CREATE TABLE. La query utilizzata nella clausola AS non può contenere alcuna colonna di tipo LONG. Durante l'operazione create table... as select, si possono disattivare le operazioni di log tramite la parola chiave NOLOGGING. Questa opzione disabilita le voci redo log che verrebbero normalmente scritte durante il riempimento della nuova tabella, migliorando così le prestazioni ma compromettendo la possibilità da parte dell'utente di recuperare questi dati nel caso si verificasse un fallimento dell'istanza prima del successivo backup.

La clausola LOB specifica la memorizzazione dei dati fuori linea per dati LOB (BLOB, CLOB e NCLOB) memorizzati internamente. Per i dettagli sulla gestione dei dati LOB si rimanda al Capitolo 32.

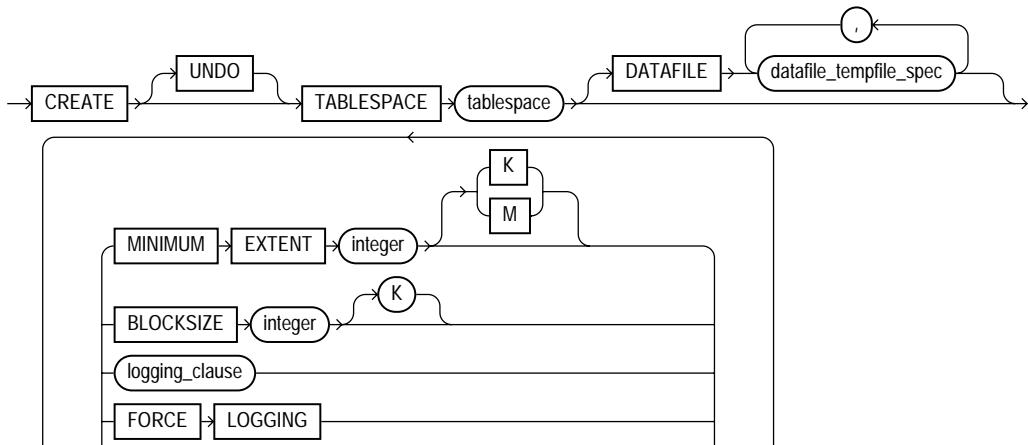
La sezione relativa alla definizione della tabella oggetto si applica alle tabelle oggetto in cui ciascuna riga contiene l'ID di un oggetto (OID). Vedere il Capitolo 33.

## CREATE TABLESPACE

**VEDERE ANCHE** ALTER TABLESPACE, DROP TABLESPACE, Capitoli 20 e 40.

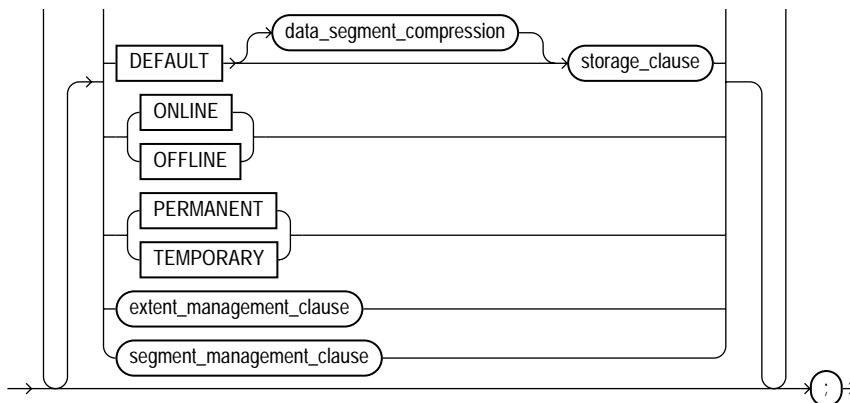
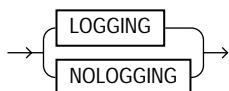
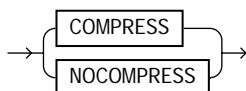
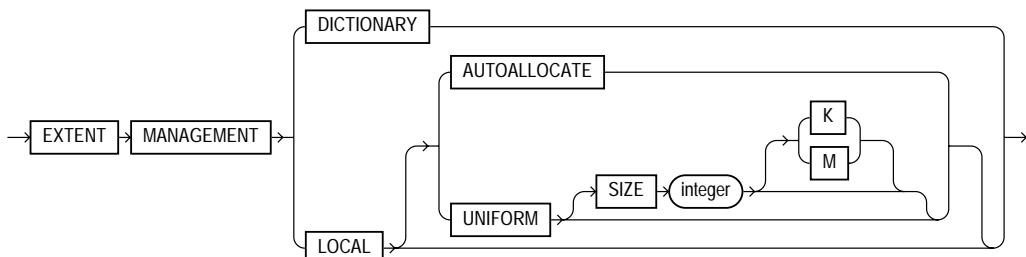
### SINTASSI

create\_tablespace



(segue)

(continua)

**logging\_clause****data\_segment\_compression****extent\_management\_clause****segment\_management\_clause**

**DESCRIZIONE** *tablespace* è il nome della tablespace e segue le convenzioni di denominazione di Oracle. **DATAFILE** è un file o una serie di file descritti secondo la *definizione\_file*, che specifica i nomi e le dimensioni dei file:

'*file*' [SIZE *intero* [*K* | *M*] [REUSE]

Il formato del file dipende dal sistema operativo. SIZE rappresenta il numero di byte riservati per questo file. Se si specifica il suffisso K, il valore viene moltiplicato per 1024; il suffisso M lo moltiplica per 1048576. DEFAULT STORAGE definisce la memorizzazione predefinita di tutti gli oggetti creati nella tablespace, a meno che questi valori predefiniti non vengano sostituiti nel comando CREATE TABLE. ONLINE, il valore predefinito, indica che questa tablespace sarà disponibile agli utenti subito dopo la sua creazione. OFFLINE impedisce l'accesso alla tablespace fino a che il comando ALTER TABLESPACE non la modifica in ONLINE. DBA\_TABLESPACES fornisce lo stato di tutte le tablespace.

SIZE e REUSE utilizzate insieme comunicano ad Oracle di riutilizzare il file se esiste già (il suo contenuto viene distrutto), oppure di crearlo se non esiste ancora. SIZE senza REUSE crea un file che non esiste ma restituisce un errore nel caso in cui il file esista già. Senza SIZE, il file deve esistere già.

Il parametro di dimensionamento MINIMUM EXTENT specifica la dimensione minima per gli extent all'interno della tablespace. Il valore predefinito è specifico per ciascun sistema operativo e dipende dalla dimensione dei blocchi del database.

Quando attivata (ON), l'opzione AUTOEXTEND estende in modo dinamico un file di dati in base alle esigenze con incrementi di dimensione pari a NEXT, fino a un massimo di MAXSIZE (o UNLIMITED, illimitato). Se una tablespace viene utilizzata solo per segmenti temporanei creati durante l'elaborazione della query, è possibile crearla come una tablespace TEMPORARY. Se invece è destinata a contenere oggetti permanenti (come le tabelle), è consigliabile utilizzare l'impostazione predefinita PERMANENT.

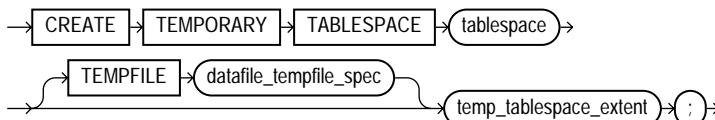
EXTENT MANAGEMENT controlla come vengono registrati i dati per la gestione degli extent per una tablespace. Per default, l'uso degli extent è LOCAL; le informazioni sulla localizzazione degli extent sono memorizzate in una bitmap all'interno dei file di dati della tablespace.

## CREATE TEMPORARY TABLESPACE

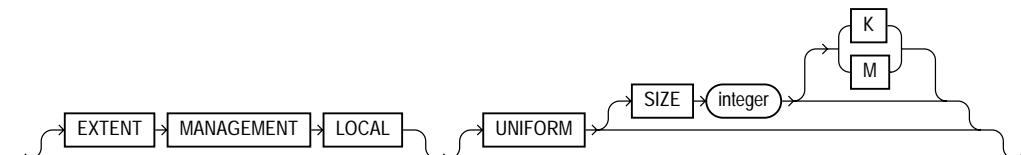
**VEDERE ANCHE** CREATE TABLE, CREATE TABLESPACE.

### SINTASSI

create\_temporary\_tablespace



temp\_tablespace\_extent



**DESCRIZIONE** CREATE TEMPORARY TABLESPACE crea una tablespace che servirà per memorizzare le tabelle temporanee. Le tabelle temporanee memorizzate in questa tablespace non sono segmenti temporanei creati durante le operazioni di ordinamento; si tratta di tabelle create con il comando `create temporary table`. Per i parametri delle tablespace si consulti la voce CREATE TABLESPACE, mentre per la sintassi delle tabelle temporanee si consulti la voce CREATE TABLE.

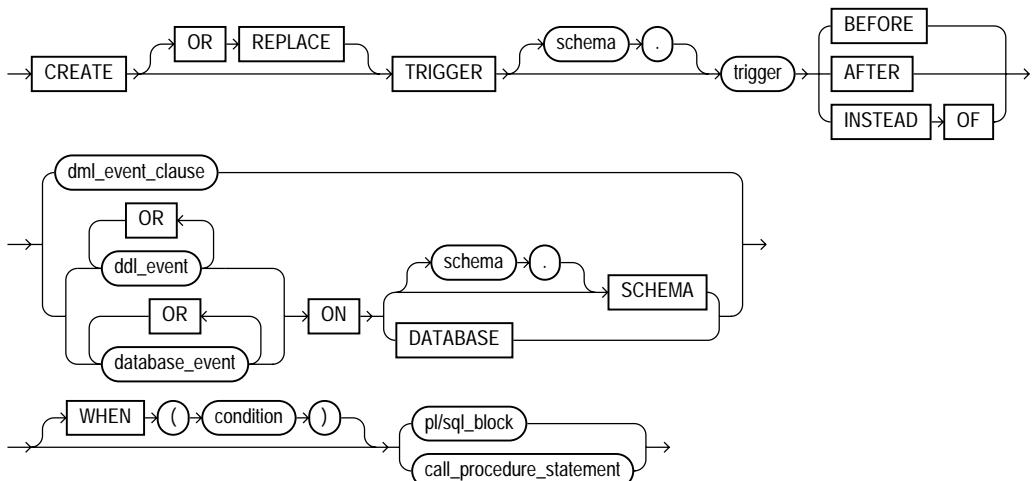
## CREATE TRIGGER

**VEDERE ANCHE** ALTER TABLE; ALTER TRIGGER; BLOCCO, STRUTTURA; DROP TRIGGER, Capitoli 28 e 30.

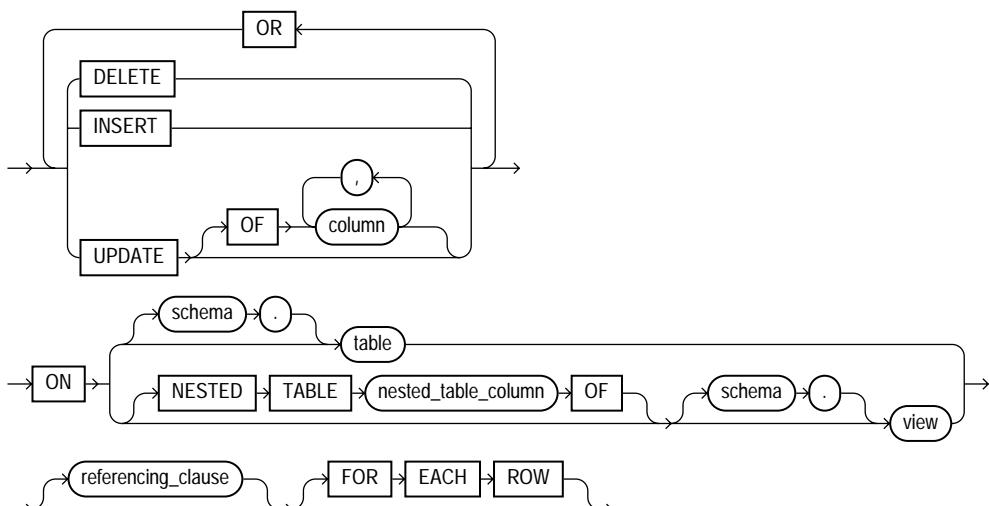
### SINTASSI

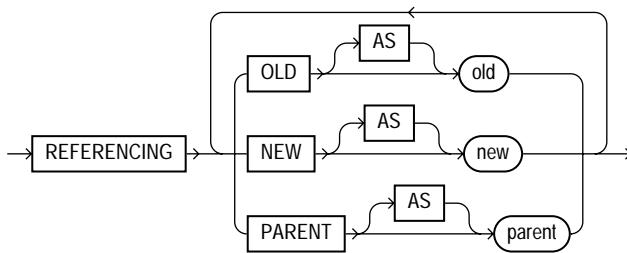
**DESCRIZIONE** CREATE TRIGGER crea e abilita un trigger per un database, ovvero il blocco di una stored procedure associata a una tabella, indicata nella clausola on, che Oracle esegue automaticamente quando l'istruzione SQL specificata viene eseguita sulla tabella. I trigger possono essere utilizzati per imporre vincoli complessi e per propagare le modifiche in tutto il database

`create_trigger`



**DML\_event\_clause**



**referencing\_clause**

anziché eseguire questa operazione nelle applicazioni. In questo modo, è sufficiente implementare il codice per il trigger una volta sola anziché doverlo ripetere per ogni applicazione.

La clausola principale del comando CREATE TRIGGER specifica l'operazione SQL che attiva il blocco (come cancellazioni, inserimenti o aggiornamenti) e il momento di attivazione del trigger prima, dopo o al posto (BEFORE, AFTER o INSTEAD OF) dell'esecuzione dell'operazione di trigger. Se si specifica una clausola OF su un trigger di tipo UPDATE, il trigger si attiva solo quando si aggiornano uno o più delle colonne specificate. Si osservi che è anche possibile creare trigger che si attivano quando si verificano eventi DDL (compresi create, alter e drop) e quando si verificano eventi di database specifici (connessioni, disconnessioni, avvii e chiusure del database ed errori del server).

La clausola REFERENCING specifica un nome di correlazione per la tabella per quanto riguarda la versione vecchia e nuova (OLD e NEW) della tabella. Ciò consente di utilizzare il nome come riferimento alle colonne per evitare confusione, in modo particolare quando il nome della tabella è OLD o NEW. I nomi predefiniti sono OLD e NEW.

La clausola FOR EACH ROW specifica un trigger di riga: il trigger si attiva una volta per ciascuna riga coinvolta da un'operazione di trigger. La clausola WHEN limita l'esecuzione del trigger in modo che avvenga solo quando la *condizione* è soddisfatta. Si tratta di una condizione SQL, non PL/SQL.

I comandi ALTER TRIGGER e ALTER TABLE consentono di disattivare e attivare i trigger. Se un trigger è disattivato, Oracle non lo attiva nel caso in cui si verifichi un'operazione potenzialmente associata a trigger. Il comando CREATE TRIGGER abilita automaticamente il trigger.

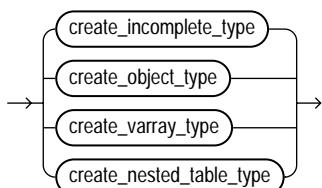
Per creare un trigger su una tabella che si possiede, è necessario disporre del privilegio di sistema CREATE TRIGGER. Per creare un trigger sulla tabella di un altro utente, occorre essere in possesso del privilegio di sistema CREATE ANY TRIGGER. Tutti i privilegi necessari all'esecuzione delle stored procedure devono essere concessi direttamente all'utente.

## **CREATE TYPE**

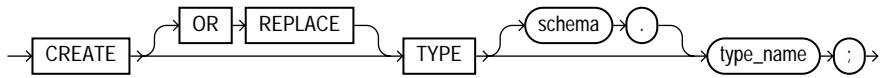
**VEDERE ANCHE** CREATE TABLE, CREATE TYPE BODY, Capitoli 4, 30 e 31.

### **SINTASSI**

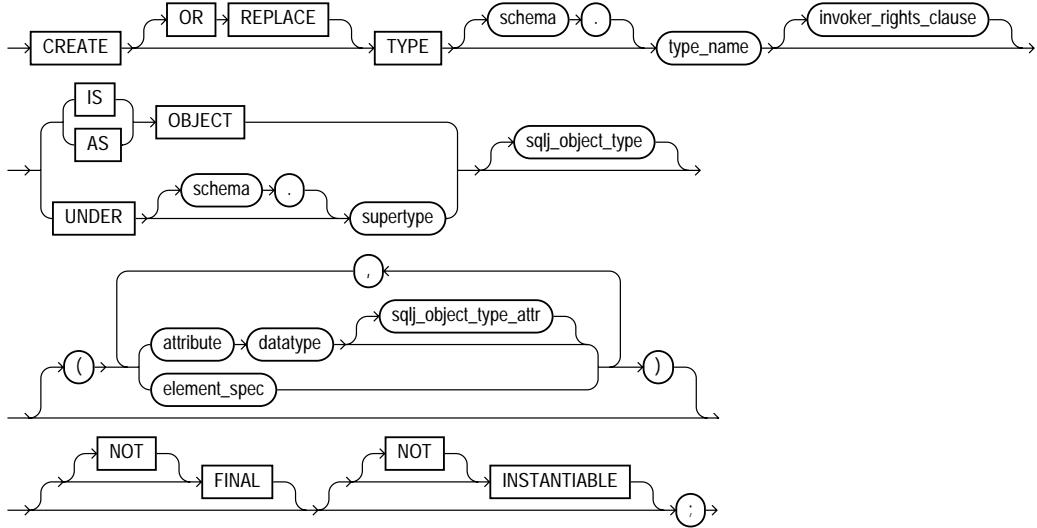
#### **create\_type**



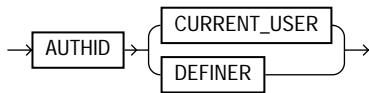
## create\_incomplete\_type



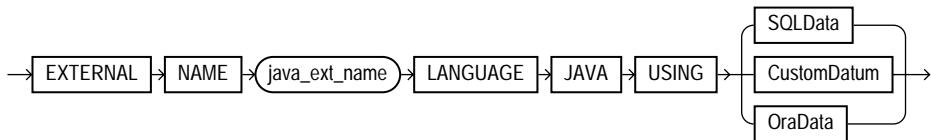
## create\_object\_type



## invoker\_rights\_clause



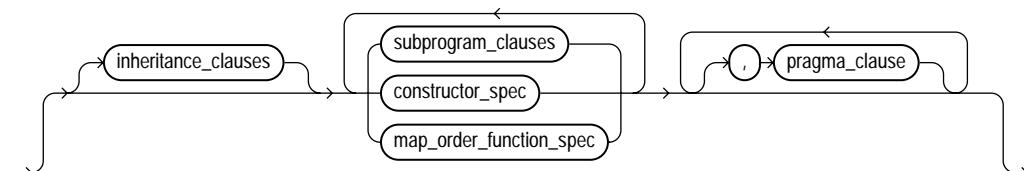
## sqlj\_object\_type

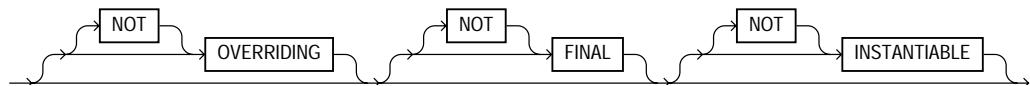
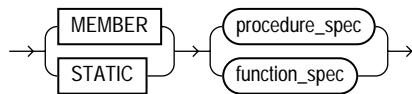
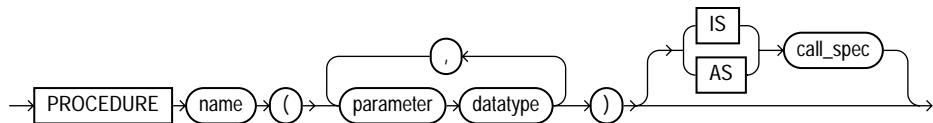
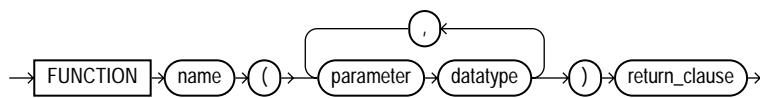
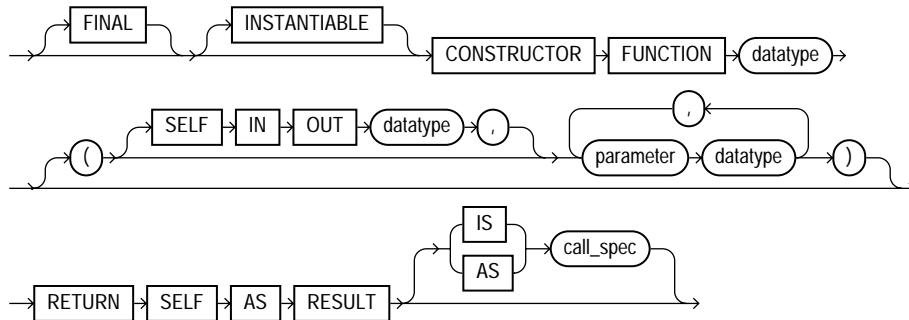
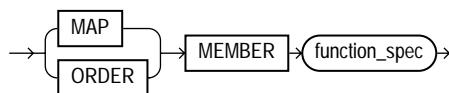
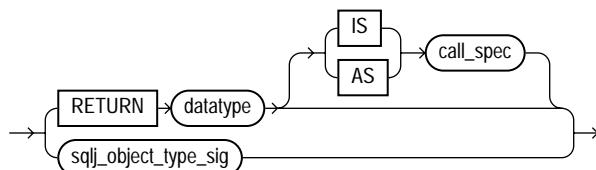


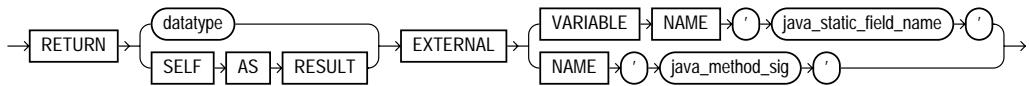
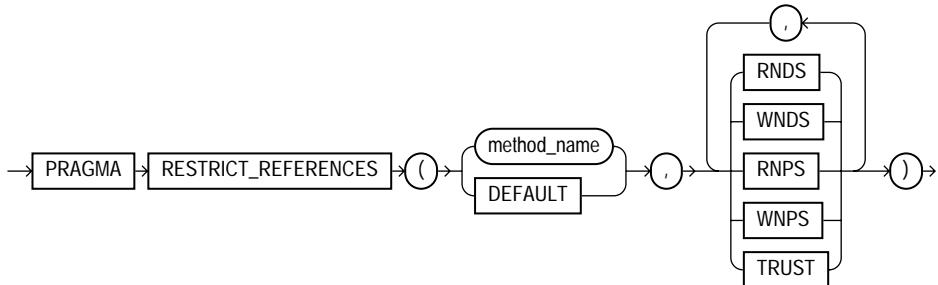
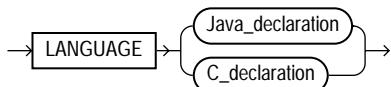
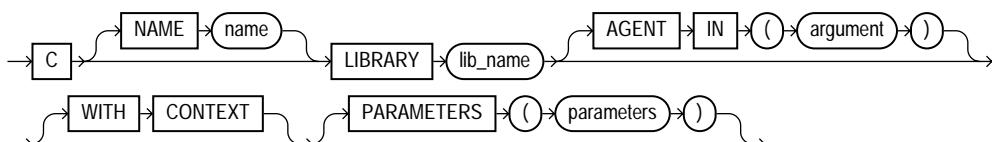
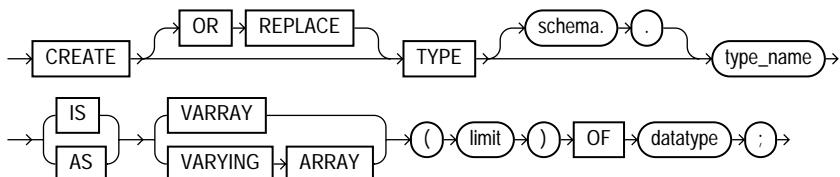
## sqlj\_object\_type\_attr

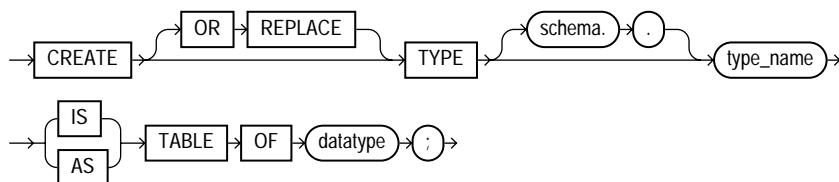


## element\_spec



**inheritance\_clauses****subprogram\_clauses****procedure\_spec****function\_spec****constructor\_spec****map\_order\_function\_spec****return\_clause**

**sqlj\_object\_type\_sig****pragma\_clause****call\_spec****Java\_declaration****C\_declaration****create\_varray\_type**

**create\_nested\_table\_type**

**DESCRIZIONE** CREATE TYPE crea un tipo di dati astratto chiamato array variabile (VARRAY), un tipo di tabella annidata oppure un tipo di oggetto incompleto. Per creare o sostituire un tipo, è necessario disporre del privilegio CREATE TYPE. Per creare un tipo nello schema di un altro utente, si deve disporre dei privilegi di sistema CREATE ANY TYPE.

Quando si crea un tipo di dati astratto, lo si può basare sui tipi di dati forniti da Oracle (come DATE e NUMBER) e sui tipi di dati astratti definiti precedentemente.

Un tipo “incompleto” possiede un nome ma non ha attributi o metodi. Tuttavia, altri tipi di oggetti possono farvi riferimento quindi, può essere utilizzato per definire tipi di oggetti che fanno riferimento l’uno all’altro. Se due tipi fanno riferimento l’uno all’altro, uno dei due dev’essere creato come tipo incompleto, quindi occorre creare il secondo tipo e infine creare nuovamente il primo con una definizione adeguata.

Se si intende creare dei metodi per il tipo, è necessario dichiarare i nomi dei metodi nella specifica del tipo. Quando si specificano i metodi, si può utilizzare la clausola PRAGMA RESTRICT\_REFERENCES per limitare la loro capacità di modificare il database. Le opzioni disponibili sono mostrate nel diagramma della sintassi. Come minimo, i metodi dovrebbero utilizzare la limitazione WNDS. Vedere il Capitolo 30. Per ulteriori informazioni sui tipi VARRAY e le tabelle annidate, si rimanda al Capitolo 31.

**ESEMPI**

Creazione di un tipo con metodi associati:

```
create or replace type ANIMALE_TY as object
(Razza      VARCHAR2(25),
Nome       VARCHAR2(25),
DataNasc   DATE,
member function ETA (DataNascita IN DATE) return NUMBER,
PRAGMA RESTRICT_REFERENCES(ETA, WNDS);
```

Creazione di un array variabile:

```
create or replace type ATTREZZI_VA as varray(5) of VARCHAR2(25);
```

Creazione di un tipo senza metodi:

```
create type INDIRIZZO_TY as object
(Via      VARCHAR2(50),
Citta    VARCHAR2(25),
Prov     CHAR(2),
Cap      NUMBER);
```

Creazione di un tipo che utilizza un altro tipo:

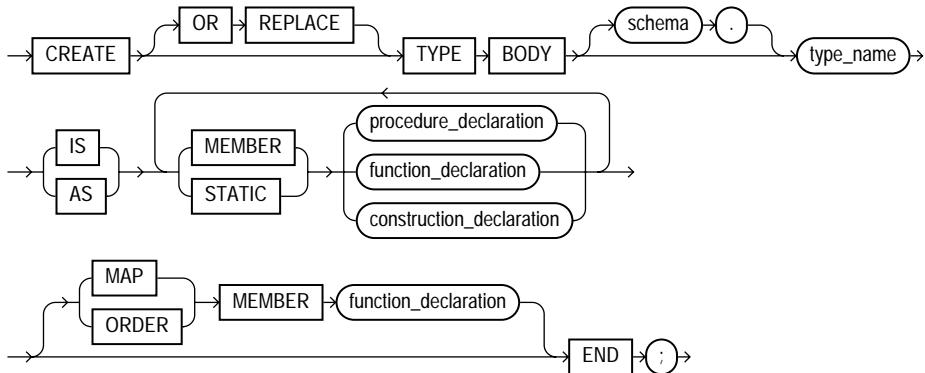
```
create type PERSONA_TY as object
(Nome      VARCHAR2(25),
Indirizzo INDIRIZZO_TY);
```

## CREATE TYPE BODY

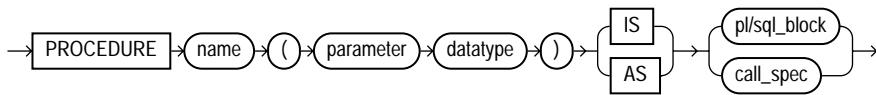
**VEDERE ANCHE** CREATE FUNCTION, CREATE PROCEDURE, CREATE TYPE, Capitoli 4, 25, 29, 30 e 31.

### SINTASSI

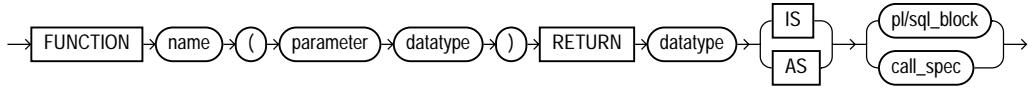
`create_type_body`



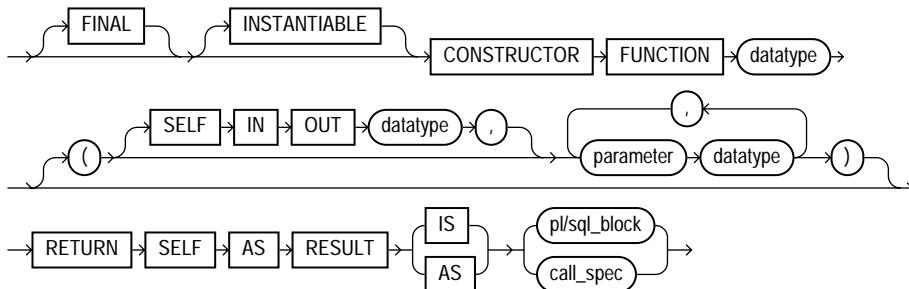
`procedure_declaration`



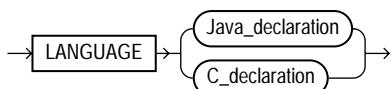
`function_declaration`

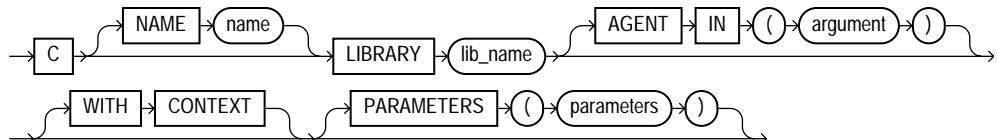


`constructor_declaration`



`call_spec`



**Java\_declaration****C\_declaration**

**DESCRIZIONE** CREATE TYPE BODY specifica i metodi da applicare ai tipi creati tramite CREATE TYPE. Per creare i corpi dei tipi, è necessario disporre del privilegio CREATE TYPE. Per creare il corpo di un tipo nello schema di un altro utente, occorre possedere il privilegio CREATE ANY TYPE.

Vedere il Capitolo 30 per esempi di metodi.

**ESEMPIO**

```

create or replace type body ANIMALE_TY as
member function Eta (DataNascita DATE) return NUMBER is
begin
    RETURN ROUND(SysDate - DataNascita);
end;
end;
  
```

**CREATE USER**

**VEDERE ANCHE** ALTER USER, CREATE PROFILE, CREATE ROLE, CREATE TABLESPACE, GRANT, Capitolo 19.

**SINTASSI**

```

CREATE USER utente IDENTIFIED
{ BY password | EXTERNALLY | GLOBALLY AS 'nome_esterno' }
[ { DEFAULT TABLESPACE tablespace
| TEMPORARY TABLESPACE tablespace
| QUOTA { intero [K | M] | UNLIMITED} ON tablespace
[ QUOTA { intero [K | M] | UNLIMITED} ON tablespace]...
| PROFILE profilo
| PASSWORD EXPIRE
| ACCOUNT { LOCK | UNLOCK } }
| DEFAULT TABLESPACE tablespace
| TEMPORARY TABLESPACE tablespace
| QUOTA { intero [K | M] | UNLIMITED} ON tablespace
[ QUOTA { intero [K | M] | UNLIMITED} ON tablespace] ...
| PROFILE profilo
| PASSWORD EXPIRE
| ACCOUNT { LOCK | UNLOCK } ]
  
```

**DESCRIZIONE** CREATE USER crea l'account di un utente che consente di accedere al database disponendo di un certo gruppo di privilegi e di impostazioni di memorizzazione. Se si specifica una password, al momento del collegamento si deve fornire questa password; se si specifica

l'opzione EXTERNALLY, l'accesso viene sottoposto a verifica attraverso le funzionalità di sicurezza del sistema operativo. La verifica di collegamenti esterni utilizza il parametro di inizializzazione OS\_AUTHENT\_PREFIX per anteporre l'ID dell'utente all'interno del sistema operativo, quindi il nome dell'utente specificato in CREATE USER deve possedere questo prefisso (solitamente OPS\$).

DEFAULT TABLESPACE è la tablespace in cui gli utenti creano gli oggetti. TEMPORARY TABLESPACE è la tablespace in cui vengono creati gli oggetti temporanei per le operazioni dell'utente.

Per entrambe le tablespace è possibile stabilire un valore QUOTA che limita la quantità di spazio, in byte (kilobyte o megabyte rispettivamente per le opzioni K o M), che un utente può allocare. La clausola profile assegna un certo profilo all'utente per limitare l'uso delle risorse del database. Se non si specifica un profilo, Oracle assegna all'utente il profilo DEFAULT.

La prima volta che si crea un utente, questo non ha privilegi. Per concedere ruoli e privilegi all'utente, è necessario il comando GRANT. Normalmente, come minimo privilegio viene concesso CREATE SESSION.

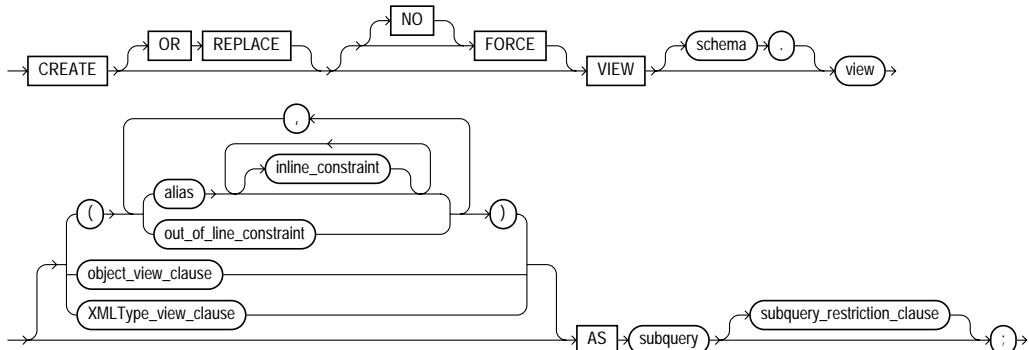
Per ulteriori informazioni sulle scadenze delle password e il blocco dell'account, si consulti la voce CREATE PROFILE e il Capitolo 19.

## CREATE VIEW

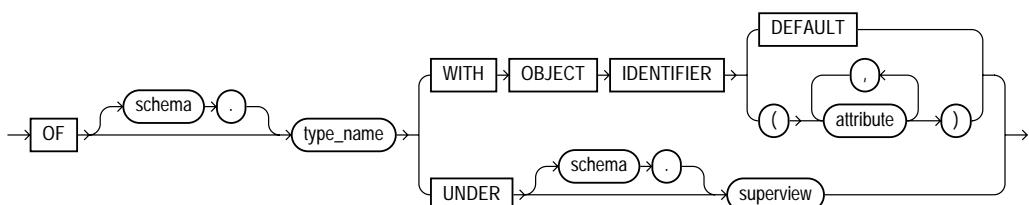
**VEDERE ANCHE** CREATE SYNONYM, CREATE TABLE, DROP VIEW, RENAME, VINCULO DI INTEGRITÀ, Capitoli 18 e 30.

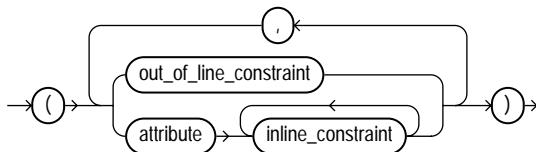
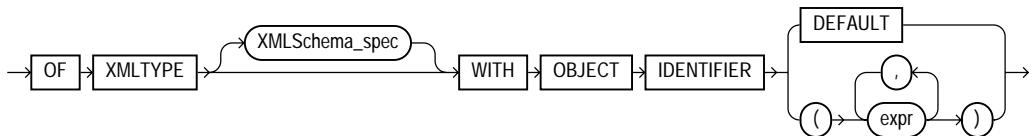
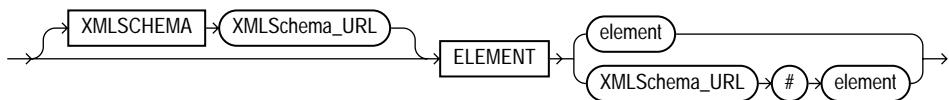
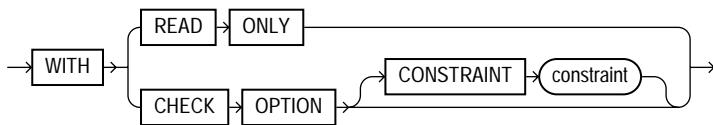
### SINTASSI

create\_view



object\_view\_clause



**object\_view\_clause2****XMLType\_view\_clause****XMLSchema\_spec****subquery\_restriction\_clause**

**DESCRIZIONE** CREATE VIEW definisce una vista di nome *vista*. *utente* è il nome dell'utente per il quale si crea la vista. L'opzione OR REPLACE ricrea la vista se questa esiste già. L'opzione FORCE crea la vista indipendentemente dal fatto che le tabelle cui essa fa riferimento esistano o meno, o se l'utente possiede i privilegi su queste tabelle. L'utente non può ancora eseguire la vista, ma la può creare. L'opzione NO FORCE crea la vista solo se le tabelle di base esistono e se l'utente possiede dei privilegi su di esse.

Se si specifica un alias, la vista lo utilizza come nome della colonna corrispondente all'interno della query. Se non si specifica alcun alias, la vista eredita il nome della colonna dalla query; in questo caso, ciascuna colonna della query deve possedere un nome univoco, conforme alle normali convenzioni di denominazione di Oracle. Non può essere un'espressione. Anche un alias presente nella query può servire per cambiare nome alla colonna.

AS query identifica le colonne delle tabelle e le altre viste destinate a comparire in questa vista. La clausola where di questa vista stabilisce quali righe debbano essere recuperate.

WITH CHECK OPTION limita gli inserimenti e gli aggiornamenti eseguiti attraverso la vista per evitare che creino righe che la vista stessa non è in grado di selezionare, in base alla clausola where dell'istruzione CREATE VIEW. WITH CHECK OPTION può essere utilizzato in una vista che si basa su un'altra vista; tuttavia, se la vista di base possiede già un'opzione WITH CHECK OPTION, viene ignorato.

*vincolo* è il nome assegnato a CHECK OPTION; si tratta di un nome facoltativo assegnato a questo vincolo. Senza di esso, Oracle assegna un nome nella forma `SYS_Cn`, dove *n* è un intero.

Un nome assegnato da Oracle in genere non rimane uguale durante le importazioni, cosa che non accade con i nomi assegnati dall'utente.

`update` e `delete` funzionano sulle righe di una vista se la vista si basa su un'unica tabella e la sua query non contiene la clausola `group by`, la clausola `distinct`, funzioni di gruppo o riferimenti alla pseudocolonna `RowNum`. Si possono aggiornare le viste contenenti altre pseudocolonne o espressioni, purché non abbiano riferimenti all'interno di `update`.

Si possono inserire righe attraverso una vista se la vista si basa su un'unica tabella e se la sua query non contiene la clausola `group by`, la clausola `distinct`, funzioni di gruppo, riferimenti a qualsiasi pseudocolonna o espressione; le righe possono essere inserite anche tramite una vista con più di una tabella usando i trigger `INSTEAD OF`. Se la vista viene creata specificando l'opzione `WITH READ ONLY`, le uniche operazioni consentite dalla vista sono le selezioni, mentre non è concessa alcuna manipolazione dei dati.

Si possono creare viste oggetto per sovrapporre tipi di dati astratti su tabelle relazionali già esistenti. Vedere il Capitolo 30.

Per creare una vista, è necessario disporre del privilegio di sistema `CREATE VIEW`. Per creare una vista nello schema di un altro utente, occorre avere il privilegio di sistema `CREATE ANY VIEW`.

## CREAZIONE DI UN DATABASE

La creazione di un database è il processo con cui si rende pronto un database per l'uso iniziale. Questa operazione comprende la pulizia dei file del database e il caricamento delle tabelle iniziali del database richieste dall'RDBMS. Tutto ciò viene effettuato attraverso l'istruzione SQL `CREATE DATABASE`.

## CTXCAT, INDICE

Un indice testuale CTXCAT, disponibile in Oracle Text, supporta le query di testo eseguite con l'operatore `CATSEARCH`. Per ulteriori informazioni su Oracle Text, si *consulti* il Capitolo 24.

## CUBE

**VEDERE ANCHE** GROUPING, ROLLUP, Capitolo 13.

### SINTASSI

`GROUP BY CUBE (colonna1, colonna2)`

**DESCRIZIONE** `CUBE` raggruppa le righe che si basano sui valori di tutte le possibili combinazioni di espressioni per ogni riga, e restituisce un'unica riga di informazioni di riepilogo per ciascun gruppo. L'operazione `CUBE` può essere utilizzata per creare report a campi incrociati. Vedere il Capitolo 13.

## CUME\_DIST

**VEDERE ANCHE** FUNZIONI DI GRUPPO.

### SINTASSI

Per gli aggregati:

---

```
CUME_DIST ( espr [, espr]... ) WITHIN GROUP
( ORDER BY
  espr [ DESC | ASC ] [ NULLS { FIRST | LAST } ]
  [, espr [ DESC | ASC ] [ NULLS { FIRST | LAST } ]]...)
```

Per le funzioni analitiche:

```
CUME_DIST ( ) OVER ( [clausola_partzisionamento_query] clausola_order_by )
```

**DESCRIZIONE** CUME\_DIST calcola la distribuzione cumulativa di un valore in un gruppo di valori. L'intervallo dei valori restituiti da CUME\_DIST va da >0 a <=1. Se si legano i valori, questi vengono sempre valutati allo stesso valore di distribuzione cumulativa.

## CURRENT\_DATE

**VEDERE ANCHE** DATE, FUNZIONI.

**SINTASSI**

```
CURRENT_DATE
```

**DESCRIZIONE** CURRENT\_DATE restituisce la data corrente nel fuso orario della sessione, con un valore del calendario Gregoriano del tipo di dati DATE.

**ESEMPIO**

```
SELECT CURRENT_DATE FROM DUAL;
```

## CURRENT\_TIMESTAMP

**VEDERE ANCHE** DATE, FUNZIONI.

**SINTASSI**

```
CURRENT_TIMESTAMP [ ( precisione ) ]
```

**DESCRIZIONE** CURRENT\_TIMESTAMP restituisce la data e l'ora correnti nel fuso orario della sessione, con un valore del tipo di dati TIMESTAMP WITH TIME ZONE. Il cambiamento del fuso orario riflette l'ora locale corrente della sessione SQL. Se si tralascia il parametro relativo alla precisione, il valore predefinito è 6. La differenza tra questa funzione e LOCALTIMESTAMP è che CURRENT\_TIMESTAMP restituisce un valore TIMESTAMP WITH TIME ZONE, mentre LOCALTIMESTAMP restituisce un valore TIMESTAMP.

*precisione* specifica la precisione al secondo frazionario del valore orario restituito.

**ESEMPIO**

```
select CURRENT_TIMESTAMP from DUAL;
```

## CURRVAL

*Vedere* PSEUDOCOLONNE.

## CURSORE

Cursore è anche un sinonimo di area di contesto, un'area di lavoro all'interno della memoria in cui Oracle memorizza l'istruzione SQL corrente. Per una query, l'area in memoria include anche le intestazioni delle colonne e una riga recuperata dalla query stessa.

## CURSORE, PL/SQL

**VEDERE ANCHE** CREATE PACKAGE, CREATE PACKAGE BODY, Capitolo 27.

### SINTASSI

```
CURSOR cursore [(parametro tipodato[,parametro tipodato]...)]
    [IS query]
```

**DESCRIZIONE** In un package PL/SQL è possibile specificare un cursore e dichiararne il corpo. La specifica contiene solo l'elenco dei parametri con i loro tipi di dati corrispondenti, non la clausola IS, mentre il corpo contiene entrambi. I parametri possono comparire in qualsiasi punto della query in cui potrebbe apparire una costante. Il cursore può essere specificato come parte delle dichiarazioni pubbliche della specifica del package, mentre il corpo del cursore può far parte del corpo del package nascosto.

## CURSORE SQL

**VEDERE ANCHE** %FOUND, %ISOPEN, %NOTFOUND, %ROWCOUNT, Capitolo 27.

**DESCRIZIONE** *SQL* è il nome del cursore implicitamente aperto ogni volta che l'istruzione SQL elaborata non fa parte di un cursore esplicitamente aperto e nominato (*vedere* DECLARE). Ne può esistere solo uno per volta. Gli attributi del cursore %FOUND e %NOTFOUND possono essere controllati verificando il valore di SQL%FOUND e SQL%NOTFOUND prima o dopo un inserimento, aggiornamento o cancellazione (che non sono mai associati a cursori esplicativi) o dopo la selezione di una singola riga eseguita senza cursori esplicativi.

Per ulteriori dettagli riguardanti queste verifiche, si consulti %FOUND. Per dettagli sui valori che questi attributi presentano in condizioni differenti, si consulti %ROWCOUNT. SQL%ISOPEN vale sempre FALSE poiché Oracle chiude il cursore SQL automaticamente dopo aver eseguito l'istruzione SQL.

## DATA CONTROL LANGUAGE (DCL), ISTRUZIONI

Le istruzioni DCL rappresentano una categoria di istruzioni SQL. Le istruzioni DCL, come grant, grant select, grant update e revoke all, controllano l'accesso ai dati e al database. Le altre categorie rappresentano istruzioni del linguaggio per la definizione dei dati (DDL) e del linguaggio per la manipolazione dei dati (DML).

## DATA DEFINITION LANGUAGE (DDL), ISTRUZIONI

Le istruzioni DDL rappresentano una categoria di istruzioni SQL. Le istruzioni DDL definiscono (tramite istruzioni create) o eliminano (tramite istruzioni drop) oggetti di database. Esempi di questo tipo di istruzioni sono create view, create table, create index, drop table, create function e rename table. L'esecuzione di qualunque comando DDL trasmette qualunque modifica in atte-

sa al database. Le altre categorie di istruzioni SQL sono quelle del linguaggio per il controllo dei dati (DCL) e del linguaggio per la manipolazione dei dati (DML).

## DATA MANIPULATION LANGUAGE (DML), ISTRUZIONI

Le istruzioni DML rappresentano una categoria di istruzioni SQL. Le istruzioni DML, quali `select`, `insert`, `delete` e `update`, consentono di interrogare e aggiornare i dati veri e propri. Le altre categorie sono istruzioni del linguaggio di controllo dei dati (DCL e del linguaggio di definizione dei dati (DDL).

## DATABASE

Database può avere una di queste due definizioni:

- un insieme di tabelle del dizionario dati e di tabelle dell’utente trattate come un’unità;
- uno o più file del sistema operativo in cui Oracle memorizza le tabelle, le viste e altri oggetti; inoltre, può essere l’insieme di oggetti del database utilizzati da una certa applicazione.

## DATABASE CHIUSO

Un database chiuso è un database associato a un’istanza (il database risulta montato) ma non aperto. I database inutilizzati vanno chiusi per ridurre il lavoro delle funzioni di gestione del database. Ciò può essere ottenuto attraverso l’istruzione SQL `ALTER DATABASE`.

## DATABASE DISTRIBUITO

Un database distribuito è una collezione di database che possono essere gestiti separatamente e condividere informazioni.

## DATABASE LINK

Un database link è un oggetto memorizzato nel database locale che identifica un database remoto, un percorso di comunicazione al database remoto e, facoltativamente, il nome di un utente e una password per questo database. Una volta definito, il database link viene utilizzato per eseguire query su tabelle nel database remoto. *Vedere il Capitolo 22.*

## DATABASE LOCALE

Un database locale è in genere un database che si trova su un computer host. *Vedere per confronto DATABASE REMOTO.*

## DATABASE, NOME

Il nome di un database è un identificatore unico utilizzato per riconoscere un database. Viene assegnato tramite il comando `create database` o nel file dei parametri di inizializzazione.

## DATABASE, OGGETTO

Un oggetto di database è qualcosa creato e memorizzato in un database. Le tabelle, le viste, i sinonimi, gli indici, le sequenze, i cluster e le colonne sono tutti tipi di oggetti del database.

## DATABASE REMOTO

Un database remoto è un database che risiede su un computer remoto. Vi si accede attraverso un database link.

## DATATYPE

*Vedere DATI, TIPI.*

## DATAZIONE GIULIANA

La datazione giuliana è un metodo di conversione per cui qualsiasi data può essere espressa come un unico numero intero. Le date giuliane possono essere ottenute utilizzando la maschera di formato 'J' nelle funzioni i cui argomenti sono date. *Vedere DATE, FORMATI.*

## DATE

DATE è uno tipo di dati standard di Oracle per memorizzare dati relativi alla data e all'ora. Il formato standard per le date è 22-APR-02. Una colonna DATE può contenere una data e un'ora comprese tra il primo gennaio del 4712 a.C. e il trentuno dicembre del 9999 d.C.

## DATE, FORMATI

**VEDERE ANCHE** DATE, FUNZIONI; Capitolo 9.

**DESCRIZIONE** I seguenti formati di date vengono utilizzati sia con TO\_CHAR che con TO\_DATE.

---

MM	Numero di mesi: 12.
RM	Numerazione romana dei mesi: XII.
MON	Abbreviazione di tre lettere del mese: AUG.
MONTH	Mese : AUGUST.
DDD	Numero del giorno nell'anno, da Jan 1: 354.
DD	Numero del giorno del mese: 23.
D	Numero del giorno nella settimana: 6.
DY	Abbreviazione del giorno a tre lettere: FRI.

---

(segue)

*(continua)*

DAY	Giorno scritto per esteso, con riempimento fino a 9 caratteri.
YYYY	Anno scritto per esteso a quattro cifre: 1946.
Y,YY	Anno, con la virgola.
YYYY	Anno con segno, 1000 B.C. = -1000.
YY	Ultime tre cifre dell'anno: 946.
Y	Ultime due cifre dell'anno: 46.
IYYY	Ultima cifra dell'anno: 6.
IYY	Standard ISO per l'anno a quattro cifre.*
IY	Standard ISO per l'anno a tre cifre.
I	Standard ISO per l'anno a due cifre.
RR	Standard ISO per l'anno a una cifra.
RRRR	Ultime due cifre dell'anno relative alla data corrente.
CC	Secolo: (20 per il 1999).
SCC	Secolo, con le date BC precedute dal segno -.
YEAR	Anno scritto per esteso: NINETEEN-FORTY-SIX.
SYEAR	Anno, con segno - prima delle date BC.
Q	Numero di trimestre: 3.
WW	Numero della settimana nell'anno, dove la settimana 1 inizia il primo giorno dell'anno.
IW	Settimane in un anno in base allo standard ISO.
W	Numero di settimane nel mese, dove la settimana 1 inizia il primo giorno del mese.
J	"Giuliano"-giorni dal 31 Dicembre, 4712 A.C.: 2422220.
HH	Ore del giorno, sempre 1-12: 11.
HH12	Identico a HH.
HH24	Ore del giorno, orologio di 24 ore: 17.
MI	Minuto dell'ora: 58.
SS	Secondo del minuto: 43.
SSSS	Secondi da mezzanotte, sempre 0-86399: 43000.
FF	Secondi frazionari come in HH.MI.SS.FF.

(segue)

(continua)

X	Carattere radice locale.
/:-.;	Punteggiatura da incorporare nella visualizzazione per TO_CHAR o ignorata nel formato per TO_DATE.
A.M.	Visualizza A.M. o P.M., a seconda dell'ora del giorno.
P.M.	Stesso effetto di A.M.
AM o PM	Stesso effetto di A.M. ma senza punti.
B.C.	Visualizza B.C. o A.D. a seconda della data.
A.D.	Identico a B.C.
BC o AD	Identico a B.C. ma senza punti.
E	Nome dell'era abbreviato, per i calendari asiatici.
EE	Nome dell'era completo, per i calendari asiatici.
TZD	Informazioni sull'ora legale.
TZH	Ora del fuso orario.
TZM	Minuto del fuso orario.
TZR	Regione del fuso orario.

\*ISO è l'International Standards Organization, che ha un set di standard per le date diverso dai formati degli Stati Uniti.

I formati di data seguenti funzionano solo con TO\_CHAR. Non funzionano con TO\_DATE.

"string"	stringa è incorporata nella visualizzazione per TO_CHAR.
Fm	Prefisso per il mese e il giorno: fmMONTH o fmday. Elimina il riempimento per il mese o il giorno (definiti in precedenza) nel formato. Senza fm, tutti i mesi vengono visualizzati con la stessa ampiezza. Stessa cosa per i giorni. Con fm, il riempimento viene eliminato. I mesi e i giorni sono di lunghezza semplicemente pari al conteggio dei caratteri che li compongono.
Fx	Formato esatto: specifica la corrispondenza di formato esatto per il modello dell'argomento carattere e per il formato della data.
TH	Suffisso per un numero: ddTH o DDTH produce 24th o 24TH. Le lettere minuscole o maiuscole dipendono da come viene scritto il numero, DD, non da come viene scritto TH. Funziona con qualsiasi numero di una data: YYYY, DD, MM, HH, MI, SS e così via.
SP	Suffisso per un numero che lo forza a essere scritto per esteso: DDSP, DdSP o ddSP produce THREE, Three o three. La scrittura maiuscola o minuscola dipende da come è scritto il numero, DD, non da come sono scritte le lettere SP. Funziona con qualsiasi numero di una data: YYYY, DD, MM, HH, MI, SS e così via.
SPTH	Suffisso combinazione di TH e SP che forza un numero a essere sia specificato per esteso che ad avere un suffisso ordinale. Mspth produce Third. La scrittura maiuscola o minuscola dipende da come è scritto il numero, DD, non da come sono scritte le lettere SP. Funziona con qualsiasi numero di una data: YYYY, DD, MM, HH, MI, SS e così via.
THSP	Identico a SPTH.

## DATE, FUNZIONI

**VEDERE ANCHE** DATA, FORMATI; Capitolo 9.

**DESCRIZIONE** Questo è un elenco alfabetico di tutte le funzioni di data correnti in SQL di Oracle. Ciascuna viene riportata altrove in questo guida, assieme alla sintassi e ad esempi d'uso.

- ADD\_MONTHS(*conta,data*) Somma *conta* mesi a *data*.
- CURRENT\_DATE Restituisce la data corrente nel fuso orario della sessione.
- CURRENT\_TIMESTAMP Restituisce la data corrente nel fuso orario della sessione.
- DBTIMEZONE Restituisce il fuso orario corrente del database, in formato UTC.
- EXTRACT (*dataora*) Estrae una porzione di una data da un valore data, come l'estrazione del valore del mese dai valori di una colonna di date.
- FROM\_TZ Converte un valore di indicatore orario in un indicatore orario con il valore di fuso orario.
- GREATEST(*data1,data2,data3,...*) Prende la più recente fra le date elencate.
- LEAST(*data1,data2,data3,...*) Prende la meno recente fra le date elencate.
- LAST\_DAY(*data*) Fornisce la data dell'ultimo giorno del mese a cui appartiene *data*.
- LOCALTIMESTAMP Restituisce l'indicatore orario nel fuso orario attivo, senza mostrare informazioni su quest'ultimo.
- MONTHS\_BETWEEN(*data2,data1*) Fornisce *data2–data1* in mesi (può essere un numero frazionario).
- NEW\_TIME(*data,'questo','altro'*) Fornisce la data (e l'ora) nel fuso orario *questo*, che dev'essere sostituito da un'abbreviazione di tre lettere per indicare il fuso orario corrente. *altro* dev'essere sostituito da un'abbreviazione di tre lettere per il fuso orario di cui si desidera conoscere la data e l'ora. I fusi orari sono i seguenti:

AST/ADT	Standard Atlantico/ora solare.
BST/BDT	Standard di Bering/ora solare.
CST/CDT	Standard Centrale/ ora solare.
EST/EDT	Standard Orientale/ora solare.
GMT	Ora di Greenwich.
HST/HDT	Standard Alaska-Hawaii/ora solare.
MST/MDT	Standard Mountain/ora solare.
NST	Tempo standard di Newfoundland.
PST/PDT	Standard Pacifico/ora solare.
YST/YDT	Standard dello Yukon/ora solare.

- NEXT\_DAY(*data,'giorno'*) fornisce la data del giorno successivo a data, dove 'giorno' è 'Lunedì', 'Martedì' e così via.
- NUMTODSINTERVAL(*n,'valore'*) Converte un numero o un'espressione *n* in un valore letterale INTERVAL DAY TO SECOND dove valore è l'unità di *n*: 'Day', 'Hour', 'Minute', 'Second'.
- NUMTOYMINTERVAL (*n,'valore'*) Converte un numero o un'espressione in un valore letterale INTERVAL YEAR TO MONTH dove valore è l'unità di *n*: 'Year', 'Month'.
- ROUND(*data,'formato'*) Senza un formato specificato arrotonda una data a 12 A.M. (mezzanotte, l'inizio del giorno) se l'ora della data è prima di mezzogiorno; altrimenti, l'arrotonda al giorno successivo. Per l'uso di un formato per l'arrotondamento, si consulti la voce "ROUND" in questo capitolo.
- SESSIONTIMEZONE Restituisce il valore del fuso orario della sessione corrente.

- **SYS\_EXTRACT\_UTC(dataora\_con\_fuso orario)** Estrae Coordinated Universal Time (UTC) da una data/ora con la differenza di fuso orario.
- **SYSTIMESTAMP** Restituisce la data di sistema, compresi i secondi frazionari e il fuso orario del database.
- **SYSDATE** Restituisce la data e ora correnti.
- **TO\_CHAR(data,'formato')** Riformatta la data secondo il formato specificato.\*
- **TO\_DATE(stringa,'formato')** Converte una stringa da un particolare formato in una data Oracle. Accetta anche un numero al posto di una stringa, con determinati limiti. ‘formato’ è ristretto.\*
- **TO\_DSINTERVAL('valore')** Converte il valore di un tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un tipo INTERVAL DAY TO SECOND.
- **TO\_TIMESTAMP('valore')** Converte il valore di un tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un valore di tipo TIMESTAMP.
- **TO\_TIMESTAMP\_TZ('valore')** Converte il valore di un tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un valore di tipo TIMESTAMP WITH TIMEZONE.
- **TO\_YMINTERVAL('valore')** Converte il valore di un tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un valore di tipo INTERVAL YEAR TO MONTH.
- **TRUNC(data,'formato')** Senza un formato imposta una data a 12 A.M. (mezzanotte, l’inizio del giorno). Per l’uso di un formato per il troncamento, si consulti la voce “TRUNC” in questo capitolo.
- **TZ\_OFFSET('valore')** Restituisce la differenza di fuso orario corrispondente al valore inserito in base alla data di esecuzione dell’istruzione.

## DATI, FILE

Un file di dati è qualsiasi file utilizzato per memorizzare dei dati in un database. Un database è composto da una o più tablespace, che a loro volta sono composte da uno o più file di dati.

## DATI, INDIPENDENZA

L’indipendenza dai dati è la proprietà di tabelle ben definite che consente di modificare la struttura fisica e logica di una tabella senza che vengano influenzate le applicazioni che vi accedono.

## DATI, TIPI

**VEDERE ANCHE** CARATTERI, FUNZIONI; CONVERSIONE, FUNZIONI; DATE, FUNZIONI; FUNZIONI DI ELENCO, NUMERI, FUNZIONI; ALTRE FUNZIONI.

**DESCRIZIONE** Quando si crea una tabella e si definiscono le colonne in essa contenute, ciascuna colonna deve appartenere a un certo tipo di dati. I tipi di dati principali di Oracle sono VARCHAR2, CHAR, DATE, LONG, LONG RAW, NUMBER, RAW e ROWID, ma per questioni di compatibilità con gli altri database SQL, le istruzioni create table di Oracle accettano diverse versioni di questi tipi. La tabella seguente elenca i tipi di dati di Oracle incorporati:

TIPO DI DATI	DEFINIZIONE
VARCHAR2( <i>dimensione</i> )	Stringa di caratteri a lunghezza variabile, con una <i>dimensione</i> massima in byte (fino a 4000).
NVARCHAR2( <i>dimensione</i> )	Stringa di caratteri a lunghezza variabile con una <i>dimensione</i> massima in byte (fino a 4000) o in caratteri, a seconda del set di caratteri nazionali scelto.
NUMBER( <i>dimensione,d</i> )	Per colonne NUMBER di <i>dimensione</i> specifica con <i>d</i> cifre dopo il punto decimale. Per esempio, NUMBER(5,2) non può contenere un valore maggiore di 999.99, altrimenti si verifica un errore.
LONG	Dati carattere a dimensione variabile fino a 2Gb di lunghezza. Per ogni tabella si può definire una sola colonna LONG. Le colonne LONG non possono essere utilizzate in subquery, funzioni, espressioni, clausole where o indici. Una tabella contenente una colonna LONG non può essere associata a cluster.
DATE	Intervallo di dati valide dal 1 Gennaio 4712 A.C. al 31 Dicembre 9999 D.C.
TIMESTAMP ( <i>precisione</i> )	Data più orario, in cui <i>precisione</i> è il numero di cifre nella parte frazionaria del campo dei secondi (il valore predefinito è 6).
TIMESTAMP ( <i>precisione</i> ) WITH TIME ZONE	TIMESTAMP più il valore relativo al cambio di fuso orario.
TIMESTAMP ( <i>precisione</i> ) WITH LOCAL TIME ZONE	TIMESTAMP, normalizzato al fuso orario locale.
INTERVAL YEAR ( <i>precisione</i> ) TO MONTH	Lasso di tempo espresso in anni e mesi, in cui <i>precisione</i> è il numero di cifre nel campo data/ora ANNO (il valore predefinito è 2).
INTERVAL DAY ( <i>precisione_giorno</i> ) TO SECOND ( <i>precisione_secondo</i> )	Lasso di tempo espresso in giorni, ore, minuti e secondi in cui <i>precisione_giorno</i> è il numero di cifre nel campo data/ora GIORNO (il valore predefinito è 2) mentre <i>precisione_secondo</i> è il numero di cifre nella parte frazionaria del campo dei secondi (il valore predefinito è 6).
RAW( <i>dimensione</i> )	Dati binari di tipo RAW, lunghi un numero di byte pari a <i>dimensione</i> . La dimensione massima è 255 byte.
LONG RAW	Dati binari di tipo raw; oppure identico a LONG.
ROWID	Valore che identifica in modo univoco una riga in un database Oracle. Viene restituito dalla pseudocolonna ROWID.
INTEGER ( <i>dimensione</i> )	Stringa esadecimale per l'indirizzo logico di una riga in una tabella di solo indice; la <i>dimensione</i> massima è di 4000 byte.
CHAR( <i>dimensione</i> )	Dati carattere a lunghezza fissa, pari a <i>dimensione</i> . La dimensione massima è 2000. Quella predefinita è di 1 byte. Riempito a destra con spazi vuoti per raggiungere la lunghezza prefissata.
NCHAR( <i>dimensione</i> )	Versione di CHAR con set di caratteri a più byte.
CLOB	Oggetto di caratteri, di grandi dimensioni, fino a 4 Gb di lunghezza.
NCLOB	Identico a CLOB, ma per set di caratteri a più byte.
BLOB	Oggetto binario di grandi dimensioni, fino a 4 Gb di lunghezza.
BFILE	Puntatore a un file binario del sistema operativo.

## DBA (AMMINISTRATORE DEL DATABASE)

Un DBA è un utente di Oracle autorizzato a concedere e revocare l'accesso al sistema per gli altri utenti, modificare le opzioni di Oracle che influiscono su tutti gli utenti ed eseguire altre funzioni amministrative. *Vedere* il Capitolo 40.

## DBTIMEZONE

**VEDERE ANCHE** DATE, FUNZIONI.

### SINTASSI

DBTIMEZONE

**DESCRIZIONE** DBTIMEZONE restituisce il valore del fuso orario del database.

### ESEMPIO

```
select DBTIMEZONE from DUAL;
```

## DBWR, PROCESSO

Uno dei processi di background utilizzati da Oracle. Il processo Database Writer scrive nuovi dati nel database. Viene definito anche come DBWn.

## DCL

*Vedere* ISTRUZIONI DEL LINGUAGGIO DI CONTROLLO DEI DATI (DCL DATA CONTROL LANGUAGE).

## DDL

*Vedere* ISTRUZIONI DEL LINGUAGGIO DI DEFINIZIONE DEI DATI (DDL DATA DEFINITION LANGUAGE).

## DDL, BLOCCO

Quando un utente esegue un comando DDL, Oracle blocca gli oggetti cui si fa riferimento nel comando con blocchi di DDL. Un blocco DDL blocca gli oggetti nel dizionario dati (si confronti questo tipo di blocco con i blocchi di analisi, ovvero gli altri tipi di blocchi nel dizionario dati).

## DEADLOCK

Un deadlock è una situazione rara che si riscontra quando due o più processi utente di un database non riescono a completare le proprie transazioni. Ciò accade perché ciascun processo trattiene una risorsa che l'altro richiede (come una riga in una tabella) per poter terminare il processo. Nonostante queste situazioni accadano di raro, Oracle rileva e risolve i deadlock eseguendo un rollback del lavoro di uno dei processi.

## DECLARE

Il comando DECLARE consente di dichiarare cursori, variabili ed eccezioni da utilizzare all'interno dei blocchi PL/SQL. Vedere il Capitolo 27.

### DECLARE CURSOR (Forma 1, Embedded SQL)

**VEDERE ANCHE** CLOSE, DECLARE DATABASE, DECLARE STATEMENT, FETCH, OPEN, PREPARE, SELECT, SELECT (Embedded SQL), guida di programmazione del precompilatore.

#### SINTASSI

```
EXEC SQL [AT {database | :variabile_host}]
DECLARE cursore CURSOR
    FOR {SELECT comando | istruzione}
```

**DESCRIZIONE** L'istruzione DECLARE CURSOR deve comparire prima di qualsiasi comando SQL che faccia riferimento a questo cursore, e dev'essere compilata nella stessa sorgente delle procedure che fanno riferimento a essa. Il suo nome dev'essere unico nella sorgente.

*database* è il nome di un database già dichiarato in un'istruzione DECLARE DATABASE e utilizzato in un comando CONNECT di SQL; *variabile\_host* è una variabile che ha come valore questo nome. *cursore* è il nome assegnato a questo cursore. SELECT *comando* è una query senza clausole INTO. In alternativa, si può utilizzare un'istruzione SQL o un blocco PL/SQL già dichiarato in un'istruzione DECLARE STATEMENT di SQL.

Quando nell'istruzione SELECT è inclusa la clausola FOR UPDATE OF, un aggiornamento può fare riferimento al cursore con where current of cursor, nonostante il cursore debba essere ancora aperto e debba essere presente una riga proveniente da un'operazione FETCH.

### DECLARE CURSOR (Forma 2, PL/SQL)

**VEDERE ANCHE** CLOSE, FETCH, OPEN, SELECT...INTO, Capitolo 27.

#### SINTASSI

```
DECLARE CURSOR cursore (parametro tipodato [.parametro tipodato]...)
IS istruzione_select
    [FOR UPDATE OF colonna [.colonna]...];
```

**DESCRIZIONE** Un cursore è un'area di lavoro che Oracle utilizza per controllare la riga attualmente elaborata. DECLARE CURSOR assegna un nome a un cursore e dichiara che esso è (IS) il risultato di una certa *istruzione\_select*. L'*istruzione\_select* è necessaria e può anche non contenere una clausola INTO. Il cursore deve essere dichiarato. A questo punto, può essere aperto (OPEN) ed è possibile eseguire l'operazione di FETCH delle righe all'interno di esso. Infine, può essere chiuso (CLOSE).

Le variabili non possono essere utilizzate direttamente nella clausola where dell'*istruzione\_select*, a meno che non siano state prima identificate come parametri, in un elenco che precede la selezione. Si osservi che DECLARE CURSOR non esegue l'istruzione select e i parametri non possiedono valori fino a che non vengono assegnati in un'istruzione OPEN. I parametri hanno nomi di oggetti standard e tipi di dati, inclusi VARCHAR2, CHAR, NUMBER, DATE e BOOLEAN, tuttavia tutti senza dimensione o scala.

FOR UPDATE OF è richiesto nel caso si desideri modificare la riga corrente all'interno del cursore utilizzando il comando update o il comando delete con la clausola CURRENT OF.

## DECLARE DATABASE (Embedded SQL)

**VEDERE ANCHE** COMMIT RELEASE (Embedded SQL), CONNECT (Embedded SQL), guida di programmazione del precompilatore.

### SINTASSI

```
EXEC SQL DECLARE database DATABASE
```

**DESCRIZIONE** DECLARE DATABASE dichiara il nome di un database remoto per utilizzarlo in successive clausole AT di istruzioni SQL, incluse COMMIT, DECLARE CURSOR, DELETE, INSERT, ROLLBACK, SELECT e UPDATE.

## DECLARE STATEMENT (Embedded SQL)

**VEDERE ANCHE** CLOSE, FETCH, OPEN, PREPARE, guida di programmazione del precompilatore.

### SINTASSI

```
EXEC SQL [AT {database | :variabile_host}]
DECLARE STATEMENT {istruzione | nome_blocco} STATEMENT
```

**DESCRIZIONE** *istruzione* è il nome dell'istruzione all'interno di DECLARE CURSOR, e dev'essere identico al nome dell'istruzione presente qui. *database* è il nome di un database precedentemente dichiarato tramite DECLARE DATABASE; *variabile\_host* può contenere come valore questo nome. Questo comando è necessario solo se DECLARE CURSOR è specificato prima di PREPARE. Quando lo si utilizza, dovrebbe essere collocato prima di DECLARE, DESCRIBE, OPEN o PREPARE e dev'essere compilato nella stessa sorgente delle procedure che vi fanno riferimento.

## DECLARE TABLE

**VEDERE ANCHE** CREATE TABLE, guida di programmazione del precompilatore.

### SINTASSI

```
EXEC SQL DECLARE tabella TABLE
  (colonna tipodato [NULL|NOT NULL],
  ...);
```

**DESCRIZIONE** *tabella* è il nome della tabella da dichiarare. *colonna* è il nome di una colonna e *tipodato* è il suo tipo di dati. Questa struttura è praticamente identica a quella di create table, compreso l'uso di NULL e NOT NULL.

Si utilizza DECLARE TABLE per comunicare ai precompilatori di ignorare le definizioni delle tabelle reali del database Oracle quando si esegue un'esecuzione con SQLCHECK=FULL. In questo caso, i precompilatori considereranno la descrizione della tabella come rilevante per il programma e ignoreranno le definizioni delle tabelle nel database. Si utilizzi questa soluzione quando le definizioni delle tabelle cambiano, oppure quando una tabella non è ancora stata creata. Se SQLCHECK non è uguale a FULL (il che significa che le tabelle e le colonne non vengono comunque controllate nel database), questo comando viene ignorato dal precompilatore e diventa semplicemente una documentazione.

## ESEMPIO

```
EXEC SQL DECLARE COMFORT TABLE (
    Citta      VARCHAR2(13) NOT NULL,
    DataCampione DATE NOT NULL,
    Mezzogiorno NUMBER(3,1),
    Mezzanotte NUMBER(3,1),
    Precipitazione NUMBER
);
```

## DECODE

**VEDERE ANCHE** ALTRE FUNZIONI, CASE, TRANSLATE, Capitolo 17.

### SINTASSI

DECODE(*valore, if1, then1[, if2, then2, ]... ,else*)

**DESCRIZIONE** *valore* rappresenta qualsiasi colonna in una tabella, a prescindere dal tipo di dati, o qualsiasi risultato di un'operazione di calcolo, quale la sottrazione di una data da un'altra, una SUBSTR di una colonna di caratteri, un numero moltiplicato un altro e così via. Per ciascuna riga viene controllato *valore*. Se è uguale a *if1*, il risultato di DECODE è uguale a *then1*; se il valore è uguale a *if2*, il risultato di DECODE risulta uguale a *then2* e così via, per tutti gli *if/then* che si riesce a costruire. Se *valore* non è uguale a nessuno degli *if*, il risultato di DECODE è *else*. Ciascuno degli *if, then e else* può essere una colonna o il risultato di una funzione o di un calcolo. Il Capitolo 16 è interamente dedicato a DECODE.

## ESEMPIO

```
select DISTINCT Citta,
    DECODE(Citta, 'SAN FRANCISCO', 'CITTA DELLA BAIA', Citta)
from COMFORT;
----- -----
CITTA      DECODE(CITTA, 'SA
KEENE      KEENE
SAN FRANCISCO Citta DELLA BAIA
```

## DECOMPOSE

**VEDERE ANCHE** CONVERSIONE, FUNZIONI.

### SINTASSI

DECOMPOSE ("stringa")

**DESCRIZIONE** DECOMPOSE traduce una stringa in qualunque tipo di dati in una stringa UNICODE dopo la sua scomposizione canonica nel medesimo set di caratteri della stringa di input.

## DEFAULT

Default è una clausola o il valore di un'opzione che viene utilizzato se non si specifica nessun'altra alternativa. Si può specificare il valore predefinito per una colonna di una tabella tramite il vincolo di colonna DEFAULT.

## **DEFINE (SQL\*PLUS)**

Vedere SET.

## **DEFINIZIONE, FASE**

La fase di definizione è una fase dell'esecuzione di una query SQL, in cui il programma definisce i buffer che devono contenere i risultati di una query da eseguire.

## **DEL**

**VEDERE ANCHE** APPEND, CHANGE, EDIT, INPUT, LIST, Capitolo 6.

### **SINTASSI**

DEL

**DESCRIZIONE** DEL elimina la linea corrente del buffer attivo in quel momento. Con un unico comando DEL si possono cancellare più righe.

### **ESEMPIO**

del

elimina la linea corrente nel buffer SQLPLUS.

Per eliminare un intervallo di linee con un unico comando, è sufficiente specificare nel comando DEL i numeri delle linee dell'inizio e della fine della sequenza. Il comando seguente cancella le linee dalla 2 alla 8 all'interno del buffer corrente.

del 2 8

## **DELETE (Forma 1, PL/SQL)**

**VEDERE ANCHE** DECLARE CURSOR, UPDATE, Capitolo 27.

**DESCRIZIONE** In PL/SQL, DELETE segue le regole di un normale comando SQL con le seguenti funzionalità aggiuntive:

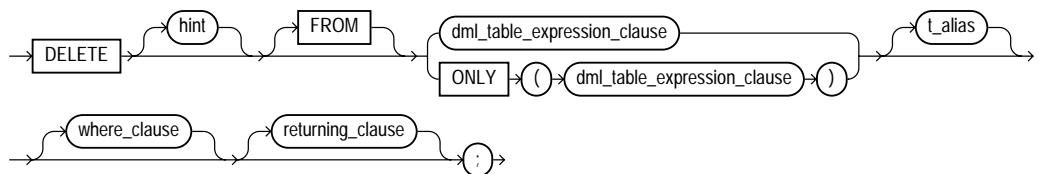
- nella clausola where, si può utilizzare una funzione e/o variabile PL/SQL proprio come una qualunque sequenza di lettere;
- DELETE . . . WHERE CURRENT OF cursor può essere utilizzato insieme a SELECT FOR UPDATE per eliminare l'ultima riga di cui è stato eseguito il FETCH. Il FETCH può essere sia esplicito che implicito da un ciclo FOR;
- come update e insert, anche il comando delete viene eseguito solo all'interno del cursore SQL. Gli attributi del cursore SQL possono essere controllati per constatare il successo del comando delete. SQL%ROWCOUNT contiene il numero di righe cancellate. Se questo valore è 0, significa che non è stata cancellata nessuna riga (inoltre, SQL%FOUND avrà come valore FALSE se nessuna è stata cancellata).

## **DELETE (Forma 2, Comando SQL)**

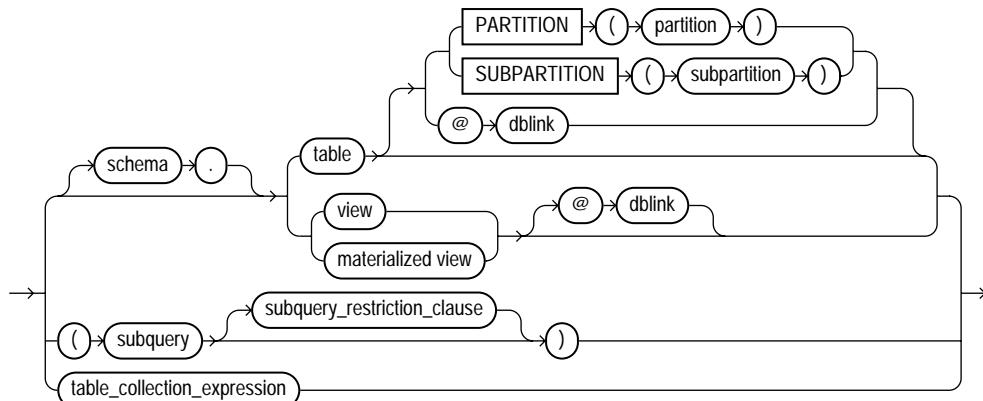
**VEDERE ANCHE** DROP TABLE, FROM, INSERT, SELECT, UPDATE, WHERE, Capitolo 15.

## SINTASSI

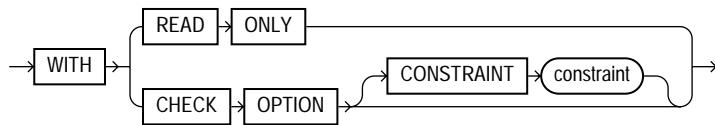
`delete`



`DML_table_expression_clause`



`subquery_restriction_clause`



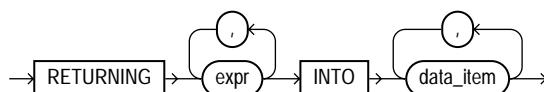
`table_collection_expression`



`where_clause`



`returning_clause`



**DESCRIZIONE** `DELETE` elimina da *tabella* tutte le righe che soddisfano *condizione*. La *condizione* può includere una query correlata e utilizzare l'alias per la correlazione. Se la tabella è

remota, si deve definire preventivamente un database link. In tal caso, `@link` ne specifica il nome. Se si specifica `link` ma non `utente`, la query cerca una tabella di proprietà dello stesso utente sul database remoto.

Quando si usa il comando `DELETE`, si può specificare una partizione o una partizione secondaria. Per cancellare le righe da una tabella, è necessario disporre dei privilegi `DELETE` sulla tabella. Se il parametro di inizializzazione `SQL92_SECURITY` è impostato a `TRUE`, si deve disporre anche dei privilegi `SELECT` se `DELETE` si riferisce a colonne nella tabella.

### ESEMPIO

Nell'esempio seguente, vengono eliminate tutte le righe della città di Keene dalla tabella `COMFORT`:

```
delete from COMFORT
where Citta = 'KEENE';
```

## **DELETE (Forma 3, Embedded SQL)**

### SINTASSI

```
EXEC SQL [AT {db_name | :variabile_host} ] [FOR :intero_host | intero]
DELETE [FROM]
[ (subquery)
| [utente.] {tabella | vista } [@blink |PARTITION (nome_partizione)] ]
[ { RETURN | RETURNING } espr [, espr]... INTO
:variabile_host [[INDICATOR] :ind_variabile]
[, :variabile_host [[INDICATOR] :ind_variable]]...]
```

**DESCRIZIONE** Questa forma del comando `DELETE` consente di eliminare righe (da una tabella, una vista o una tabella di solo indice) dall'interno di un programma di precompilatore.

## **DENSE\_RANK**

**VEDERE ANCHE** FUNZIONI DI GRUPPO, Capitolo 13.

### SINTASSI

```
DENSE_RANK ( espr [, espr]... ) WITHIN GROUP
( ORDER BY
espr [ DESC | ASC ] [NULLS { FIRST | LAST }]
[, espr [ DESC | ASC ] [NULLS { FIRST | LAST }]]...)
```

Per le funzioni analitiche:

```
DENSE_RANK ( ) OVER ( [clausola_partizionamento_query] clausola_order_by )
```

**DESCRIZIONE** `DENSE_RANK` calcola l'ordine di una riga in un gruppo ordinato di righe. Per esempi correlati, si *consulti* il Capitolo 13.

## **DEREF**

L'operatore `DEREF` restituisce il valore di un oggetto di riferimento. *Vedere* il Capitolo 31.

## **DESCRIBE (Forma 1, Comando SQL\*PLUS)**

**VEDERE ANCHE** CREATE TABLE.

### **SINTASSI**

DESC[RIBE] [*utente*.]*tabella*

**DESCRIZIONE** DESCRIBE visualizza la definizione della tabella specificata. Se si omette *utente*, SQL\*PLUS visualizza la tabella di proprietà di chi ha eseguito il comando. La definizione include la tabella e le sue colonne, con il nome di ciascuna colonna, lo stato NULL o NOT NULL, il tipo di dati e l'ampiezza o precisione. DESCRIBE può essere utilizzato anche sulle viste, sui sinonimi o sulle specifiche di funzioni e procedure.

### **ESEMPIO**

describe COMFORT

Nome	Null?	Type
CITTA	NOT NULL	VARCHAR2(13)
DATACAMPIONE	NOT NULL	DATE
MEZZOGIORNO		NUMBER(3,1)
MEZZANOTTE		NUMBER(3,1)
PRECIPITAZIONE		NUMBER

## **DESCRIBE (Forma 2, Embedded SQL)**

**VEDERE ANCHE** PREPARE.

### **SINTASSI**

```
EXEC SQL DESCRIBE [ BIND VARIABLES FOR
                     | SELECT LIST FOR]
                     nome_istruzione INTO descrittore
```

**DESCRIZIONE** Il comando DESCRIBE inizializza un descrittore in modo che contenga le descrizioni delle variabili host per un'istruzione SQL dinamica o un blocco PL/SQL. Il comando DESCRIBE segue il comando PREPARE.

## **DIFFERITO, SEGMENTO DI ROLLBACK**

Un segmento di rollback differito è un segmento contenente voci che non possono essere applicate alla tablespace, perché quest'ultima è scollegata. Non appena la tablespace ritorna a essere disponibile, tutte le voci vengono applicate.

## **DIMENSIONE**

Una dimensione è un insieme specifico di gerarchie tra le colonne delle tabelle. Per esempio, nella dimensione geografia, i Paesi fanno parte dei continenti e gli Stati fanno parte dei Paesi. Quando si definiscono le relazioni tra le dimensioni, Oracle potrà utilizzare queste informazioni quando valuta le opzioni dell'ottimizzatore. Se esistono viste materializzate che contengono dati

aggregati, l'ottimizzatore potrà usare le definizioni delle dimensioni per stabilire se è possibile accedere alla vista materializzata invece che alla tabella di base. Vedere CREATE DIMENSION.

## DIRECTORY

Un nome logico che punta a una directory fisica, utilizzata dai LOB di BFILE e dalle tabelle esterne. Vedere CREATE DIRECTORY e il Capitolo 25.

## DISASSOCIATE STATISTICS

**VEDERE ANCHE** ASSOCIATE STATISTICS.

### SINTASSI

```
DISASSOCIATE STATISTICS FROM
{ COLUMNS [schema .] tabella . colonna [, [schema .] tabella . colonna]...
| FUNCTIONS [schema .] funzione [, [schema .] funzione]...
| PACKAGES [schema .] package [, [schema .] package]...
| TYPES [schema .] tipo [, [schema .] tipo]...
| INDEXES [schema .] indice [, [schema .] indice]...
| INDEXTYPES [schema .] tipoindice [. [schema .] tipoindice]... }
[FORCE];
```

**DESCRIZIONE** ASSOCIATE STATISTICS associa un set di funzioni statistiche con una o più colonne, funzioni standalone, package, tipi o indici. DISASSOCIATE STATISTICS revoca le associazioni. Si deve disporre del privilegio ALTER sull'oggetto di base.

## DISCONNECT

**VEDERE ANCHE** CONNECT, EXIT, QUIT.

### SINTASSI

```
DISC[ONNECT]
```

**DESCRIZIONE** DISCONNECT esegue il commit delle modifiche pendenti nel database e si scollega da Oracle. La sessione SQL\*PLUS dell'utente resta attiva e le funzionalità di SQL\*PLUS continuano a funzionare, ma senza alcuna connessione al database. Si possono modificare i contenuti dei buffer, utilizzare il proprio editor, eseguire lo spooling o fermarlo, oppure ricollegarsi a Oracle, ma fino a che la connessione non è stabilita, nessuna istruzione SQL può essere eseguita.

EXIT o QUIT riportano l'utente all'interno del proprio sistema operativo. Se si è ancora connessi quando si esegue EXIT o QUIT, si verrà scollegati automaticamente e si dovrà uscire da Oracle.

## DISTINCT

Distinct significa unico. Viene utilizzato all'interno di costrutti select e funzioni di gruppo. Vedere le parti relative alle FUNZIONI DI GRUPPO e al costrutto SELECT.

## DIZIONARIO, BLOCCHI

Un blocco relativo a un dizionario è un blocco condiviso posseduto da utenti che analizzano istruzioni DML, oppure un blocco esclusivo di proprietà degli utenti che eseguono comandi DDL, per evitare che una tabella venga alterata mentre si eseguono query sui dizionari. Possono esistere più blocchi di questo tipo contemporaneamente.

## DIZIONARIO, CACHE

La cache del dizionario memorizza le informazioni del dizionario dati nella SGA. Le informazioni del dizionario presenti nella cache migliorano le prestazioni, in quanto le informazioni del dizionario vengono utilizzate di frequente. La cache del dizionario dati fa parte dello Shared SQL Pool.

## DIZIONARIO DATI

Il dizionario dati rappresenta un insieme completo delle tabelle e delle viste di proprietà degli utenti SYS e SYSTEM del DBA, che viene attivato la prima volta che si installa Oracle ed è una fonte centrale di informazioni per gli RDBMS Oracle e per tutti gli utenti. Le tabelle vengono gestite automaticamente da Oracle e includono un gruppo di viste e tabelle contenenti informazioni relative agli oggetti, agli utenti, ai privilegi, agli eventi e all'utilizzo del database. *Vedere il Capitolo 37.*

## DML

*Vedere ISTRUZIONI DEL LINGUAGGIO DI MANIPOLAZIONE DEI DATI (DML).*

## DML LOCK

Quando un utente esegue un'istruzione SQL, i dati cui fa riferimento l'istruzione vengono bloccati in una certa modalità. L'utente può anche bloccare i dati in modo esplicito con un'istruzione LOCK.

## DROP CLUSTER

**VEDERE ANCHE** CREATE CLUSTER, DROP TABLE, Capitolo 20.

### SINTASSI

```
DROP CLUSTER [schema .] cluster
[INCLUDING TABLES [CASCADE CONSTRAINTS]];
```

**DESCRIZIONE** DROP CLUSTER elimina un cluster dal database. Se il cluster non fa parte del proprio schema, è necessario possedere il privilegio DROP ANY CLUSTER. Non è possibile eliminare un cluster che contiene delle tabelle. Per prima cosa, è necessario eliminare le tabelle. La clausola INCLUDING TABLES cancella automaticamente tutte le tabelle contenute all'interno dei cluster. L'opzione CASCADE CONSTRAINTS elimina tutti i vincoli di integrità referenziale

dalle tabelle esterne al cluster che però fanno riferimento alle chiavi contenute nelle tabelle raccolte in cluster.

Non è possibile rimuovere da un cluster le singole tabelle. Per ottenere lo stesso effetto, è necessario copiare la tabella con un nome nuovo (si utilizzi CREATE TABLE assieme ad AS SELECT), eliminare la tabella vecchia (rimuovendola così dal cluster), assegnare alla copia lo stesso nome della tabella appena eliminata (mediante l'istruzione RENAME) quindi concedere i privilegi opportuni e creare gli indici necessari.

## DROP CONTEXT

**VEDERE ANCHE** CREATE CONTEXT.

### SINTASSI

```
DROP CONTEXT spazionomi;
```

**DESCRIZIONE** DROP CONTEXT elimina uno spazio di nomi contestuale dal database. Si deve disporre del privilegio di sistema DROP ANY CONTEXT.

## DROP DATABASE LINK

**VEDERE ANCHE** CREATE DATABASE LINK, Capitolo 22.

### SINTASSI

```
DROP [PUBLIC] DATABASE LINK collegamento;
```

**DESCRIZIONE** DROP DATABASE LINK elimina un database link di proprietà dell'utente. Nel caso si tratti di un collegamento pubblico, è necessario utilizzare la parola chiave facoltativa PUBLIC, e per usarla occorre essere un DBA. PUBLIC non può essere utilizzata per eliminare un collegamento privato. *collegamento* è il nome del collegamento da eliminare. Per poter eliminare un database link pubblico, occorre disporre del privilegio di sistema DROP PUBLIC DATABASE LINK. All'interno del proprio account è possibile eliminare database link privati.

### ESEMPIO

La linea seguente elimina un database link di nome SEDE\_DI\_ANA:

```
drop database link SEDE_DI_ANA;
```

## DROP DIMENSION

**VEDERE ANCHE** CREATE DIMENSION.

### SINTASSI

```
DROP DIMENSION [schema .] dimensione;
```

**DESCRIZIONE** DROP DIMENSION elimina una dimensione di proprietà dell'utente. Si deve disporre del privilegio di sistema DROP ANY DIMENSION.

### ESEMPIO

La linea seguente elimina la dimensione GEOGRAFIA:

```
drop dimension GEOGRAFIA;
```

## DROP DIRECTORY

**VEDERE ANCHE** CREATE DIRECTORY, Capitoli 25, 32.

### SINTASSI

`DROP DIRECTORY nome_directory;`

**DESCRIZIONE** DROP DIRECTORY elimina una directory già esistente. Per eliminare una directory è indispensabile avere il privilegio DROP ANY DIRECTORY.

**NOTA** *Non si deve eliminare una directory mentre i suoi file sono in uso.*

### ESEMPIO

La linea seguente elimina la directory di nome RECENSIONI:

```
drop directory RECENSIONI;
```

## DROP FUNCTION

**VEDERE ANCHE** ALTER FUNCTION, CREATE FUNCTION, Capitolo 29.

### SINTASSI

`DROP FUNCTION [schema .] nome_funzione;`

**DESCRIZIONE** DROP FUNCTION elimina la funzione specificata. Oracle invalida qualsiasi oggetto che dipenda o che chiama la funzione in questione. Per eliminare una funzione che non si possiede, occorre avere il privilegio di sistema DROP ANY PROCEDURE.

## DROP INDEX

**VEDERE ANCHE** ALTER INDEX, CREATE INDEX, CREATE TABLE, Capitolo 20.

### SINTASSI

`DROP INDEX [schema .] indice [FORCE];`

**DESCRIZIONE** DROP INDEX elimina l'indice specificato. È necessario essere i proprietari dell'indice specificato o avere il privilegio di sistema DROP ANY INDEX. FORCE si applica soltanto agli indici di dominio. Questa clausola elimina l'indice di dominio anche se l'invocazione della routine del tipo di indice restituisce un errore oppure l'indice è contrassegnato come IN PROGRESS.

## DROP INDEXTYPE

**VEDERE ANCHE** CREATE INDEXTYPE.

### SINTASSI

`DROP INDEXTYPE [schema .] tipoindice [FORCE];`

**DESCRIZIONE** DROP INDEXTYPE elimina il tipo di indice specificato e qualsiasi associazione con tipi di statistiche. Si deve possedere il privilegio di sistema DROP ANY INDEXTYPE.

## DROP JAVA

**VEDERE ANCHE** CREATE JAVA, Capitolo 36.

### SINTASSI

```
DROP JAVA { SOURCE | CLASS | RESOURCE } [schema .] nome_oggetto;
```

**DESCRIZIONE** DROP JAVA elimina la classe, la sorgente o la risorsa Java specificata. Si deve disporre del privilegio di sistema DROP PROCEDURE. Se l'oggetto si trova nello schema di un altro utente, si deve possedere il privilegio di sistema DROP ANY PROCEDURE.

## DROP LIBRARY

**VEDERE ANCHE** CREATE LIBRARY, Capitolo 29.

### SINTASSI

```
DROP LIBRARY nome_libreria;
```

**DESCRIZIONE** DROP LIBRARY elimina la libreria specificata. Si deve possedere il privilegio di sistema DROP ANY LIBRARY.

## DROP MATERIALIZED VIEW

**VEDERE ANCHE** ALTER MATERIALIZED VIEW, CREATE MATERIALIZED VIEW, Capitolo 23.

### SINTASSI

```
DROP MATERIALIZED VIEW [schema .] vista_materializzata;
```

**DESCRIZIONE** DROP MATERIALIZED VIEW elimina la vista materializzata indicata. Si deve possedere la vista materializzata oppure disporre del privilegio di sistema DROP ANY SNAPSHOT o DROP ANY MATERIALIZED VIEW.

La parola chiave SNAPSHOT è accettata in sostituzione di MATERIALIZED VIEW per garantire la compatibilità con le versioni precedenti.

## DROP MATERIALIZED VIEW LOG

**VEDERE ANCHE** ALTER MATERIALIZED VIEW LOG, CREATE MATERIALIZED VIEW LOG, Capitolo 23.

### SINTASSI

```
DROP MATERIALIZED VIEW LOG ON [schema .] tabella;
```

**DESCRIZIONE** DROP MATERIALIZED VIEW LOG elimina la tabella di log indicata. Per eliminare il log di una vista materializzata si deve disporre dei privilegi necessari per eliminare una tabella. Dopo l'eliminazione del log, qualsiasi vista materializzata sulla tabella master verrà aggiornata completamente, ma non tramite aggiornamenti rapidi.

La parola chiave SNAPSHOT è accettata in sostituzione di MATERIALIZED VIEW per garantire la compatibilità con le versioni precedenti.

## DROP OPERATOR

**VEDERE ANCHE** ALTER OPERATOR.

### SINTASSI

DROP OPERATOR [*schema .*] *operatore* [FORCE];

**DESCRIZIONE** DROP OPERATOR elimina un operatore definito dall'utente. È necessario essere proprietari dell'operatore oppure possedere il privilegio di sistema DROP ANY OPERATOR. Si specifichi FORCE per eliminare l'operatore anche se attualmente vi fanno riferimento uno o più oggetti di schema (tipi di indice, package, funzioni, procedure e così via) e si contrassegnino questi oggetti dipendenti come INVALID.

## DROP OUTLINE

**VEDERE ANCHE** CREATE OUTLINE.

### SINTASSI

DROP OUTLINE *struttura*;

**DESCRIZIONE** DROP OUTLINE elimina una struttura memorizzata dal database. Si deve possedere il privilegio di sistema DROP ANY OUTLINE. I percorsi esecutivi delle esecuzioni successive di istruzioni SQL, che utilizzavano questa struttura, verranno valutati al momento dell'esecuzione.

## DROP PACKAGE

**VEDERE ANCHE** ALTER PACKAGE CREATE PACKAGE, Capitolo 29.

### SINTASSI

DROP PACKAGE [BODY] [*schema .*]*package*

**DESCRIZIONE** DROP PACKAGE elimina il package specificato. Se si utilizza la clausola facoltativa BODY, si elimina il solo corpo senza intaccare la specifica del package. Oracle invalida qualsiasi oggetto che dipenda dal package se si è scelto di cancellare la specifica del package; lo stesso non vale se invece si elimina solamente il corpo. È necessario essere proprietari del package o possedere il privilegio di sistema DROP ANY PROCEDURE.

## DROP PROCEDURE

**VEDERE ANCHE** ALTER PROCEDURE, CREATE PROCEDURE, Capitolo 29.

### SINTASSI

DROP PROCEDURE [*schema .*] *procedura*;

**DESCRIZIONE** DROP PROCEDURE elimina la procedura specificata. Oracle invalida qualsiasi oggetto che dipenda o che chiama la procedura in questione. È necessario essere proprietari della procedura o possedere il privilegio di sistema DROP ANY PROCEDURE.

## DROP PROFILE

**VEDERE ANCHE** ALTER PROFILE, CREATE PROFILE, Capitolo 19.

### SINTASSI

```
DROP PROFILE profilo [CASCADE];
```

**DESCRIZIONE** DROP PROFILE elimina il profilo specificato. Si deve avere il privilegio di sistema DROP PROFILE.

## DROP ROLE

**VEDERE ANCHE** ALTER ROLE, CREATE ROLE, Capitolo 19.

### SINTASSI

```
DROP ROLE ruolo;
```

**DESCRIZIONE** DROP ROLE elimina il ruolo specificato. Si deve disporre del ruolo WITH ADMIN OPTION oppure essere in possesso del privilegio di sistema DROP ANY ROLE.

## DROP ROLLBACK SEGMENT

**VEDERE ANCHE** ALTER ROLLBACK SEGMENT, CREATE ROLLBACK SEGMENT, CREATE TABLESPACE, SHUTDOWN, STARTUP, Capitolo 40.

### SINTASSI

```
DROP ROLLBACK SEGMENT segmento_rollback;
```

**DESCRIZIONE** *segmento\_rollback* è il nome di un segmento di rollback già esistente da eliminare. Il segmento non dev'essere attivo mentre si esegue l'istruzione. L'opzione PUBLIC è necessaria quando si devono eliminare segmenti di rollback pubblici.

La colonna Status della vista del dizionario dati DBA\_ROLLBACK\_SEGS può mostrare quali segmenti di rollback sono in uso. Se il segmento è attualmente in uso, si può attendere fino a che venga rilasciato o chiudere il database con l'opzione IMMEDIATE per poi aprirlo in modalità esclusiva con STARTUP. Per eliminare un segmento di rollback, è necessario avere il privilegio di sistema DROP\_ROLLBACK\_SEGMENT.

## DROP SEQUENCE

**VEDERE ANCHE** ALTER SEQUENCE, CREATE SEQUENCE, Capitolo 20.

### SINTASSI

```
DROP SEQUENCE [schema .] nome_sequenza;
```

**DESCRIZIONE** *nome\_sequenza* è il nome della sequenza da eliminare. Per effettuare questa operazione, si deve possedere la sequenza in questione oppure disporre del privilegio di sistema DROP ANY SEQUENCE.

## DROP SNAPSHOT

*Vedere* DROP MATERIALIZED VIEW.

## DROP SNAPSHOT LOG

*Vedere* DROP MATERIALIZED VIEW LOG.

## DROP SYNONYM

**VEDERE ANCHE** CREATE SYNONYM, Capitolo 22.

### SINTASSI

`DROP [PUBLIC] SYNONYM [schema .] sinonimo;`

**DESCRIZIONE** DROP SYNONYM elimina il sinonimo specificato. Per eliminare un sinonimo pubblico, è necessario possedere il privilegio di sistema DROP ANY PUBLIC SYNONYM. Per eliminare un sinonimo privato, occorre esserne il proprietario o possedere il privilegio di sistema DROP ANY SYNONYM.

## DROP TABLE

**VEDERE ANCHE** ALTER TABLE, CREATE INDEX, CREATE TABLE, DROP CLUSTER, Capitolo 18.

### SINTASSI

`DROP TABLE [schema .] tabella [CASCADE CONSTRAINTS]`

**DESCRIZIONE** DROP TABLE elimina la tabella specificata. Per eliminare una tabella è necessario esserne il proprietario o possedere il privilegio di sistema DROP ANY TABLE. Quando viene eliminata una tabella, di fatto vengono soppressi anche tutti gli indici e le concessioni a essa associati. Gli oggetti contenuti in tabelle in corso di cancellazione vengono marcati come non validi e cessano di funzionare.

L'opzione CASCADE CONSTRAINTS elimina tutti i vincoli di integrità referenziale che fanno riferimento alle chiavi della tabella in corso di cancellazione.

È possibile eliminare un cluster con tutte le tabelle relative utilizzando la clausola INCLUDING TABLES del comando DROP CLUSTER.

## DROP TABLESPACE

**VEDERE ANCHE** ALTER TABLESPACE, CREATE DATABASE, CREATE TABLESPACE, Capitoli 20 e 40.

### SINTASSI

`DROP TABLESPACE tablespace  
[INCLUDING CONTENTS [AND DATAFILES] [CASCADE CONSTRAINTS]];`

**DESCRIZIONE** *tablespace* è il nome della tablespace da eliminare. L'opzione INCLUDING CONTENTS consente di eliminare una tablespace anche se contiene dei dati. Senza specificare quest'ultima opzione possono essere cancellate solo tablespace vuote. Prima della cancellazione, le tablespace devono essere fuori linea (*vedere* ALTER TABLESPACE), altrimenti la cancellazione potrebbe essere impedita da utenti che accedono ai dati, agli indici, ai segmenti temporanei o di rollback contenuti nella tablespace. Per utilizzare questo comando, si deve disporre del privilegio di sistema DROP TABLESPACE.

## DROP TRIGGER

**VEDERE ANCHE** ALTER TRIGGER, CREATE TRIGGER, Capitoli 28 e 30.

### SINTASSI

```
DROP TRIGGER [schema .] trigger;
```

**DESCRIZIONE** DROP TRIGGER elimina il trigger specificato. Si deve possedere il trigger oppure disporre del privilegio di sistema DROP ANY TRIGGER.

## DROP TYPE

**VEDERE ANCHE** CREATE TYPE, Capitoli 4 e 30.

### SINTASSI

```
DROP TYPE [schema .] nome_tipo [ FORCE | VALIDATE ];
```

**DESCRIZIONE** DROP TYPE elimina la specifica e il corpo di un tipo di dati astratto. Si deve possedere il tipo o disporre del privilegio di sistema DROP ANY TYPE. Non è possibile eliminare un tipo se una tabella o qualsiasi altro oggetto vi sta ancora facendo riferimento.

## DROP TYPE BODY

**VEDERE ANCHE** CREATE TYPE, CREATE TYPE BODY, DROP TYPE Capitolo 30.

### SINTASSI

```
DROP TYPE BODY [schema .] nome_tipo;
```

**DESCRIZIONE** DROP TYPE BODY elimina il corpo del tipo di dati specificato. È necessario esserne il proprietario o disporre del privilegio di sistema DROP ANY TYPE.

## DROP USER

**VEDERE ANCHE** ALTER USER, CREATE USER, Capitolo 19.

### SINTASSI

```
DROP USER utente [CASCADE];
```

**DESCRIZIONE** DROP USER elimina l'utente specificato. Si deve disporre del privilegio di sistema DROP USER. L'opzione CASCADE elimina tutti gli oggetti nello schema dell'utente prima di cancellare l'utente. È indispensabile utilizzare l'opzione CASCADE se l'utente ha ancora degli oggetti all'interno del proprio schema.

## DROP VIEW

**VEDERE ANCHE** CREATE SYNONYM, CREATE TABLE, CREATE VIEW, Capitolo 18.

### SINTASSI

```
DROP VIEW [schema .] vista [CASCADE CONSTRAINTS];
```

**DESCRIZIONE** DROP VIEW elimina la vista specificata. La vista da eliminare deve essere di proprietà dell'utente, oppure questi deve possedere il privilegio di sistema DROP ANY VIEW.

## DROPJAVA

DROPJAVA è una utility che serve per rimuovere le classi Java dal database. *Vedere* LOADJAVA.

## DUAL

**VEDERE ANCHE** FROM, SELECT, Capitolo 9.

**DESCRIZIONE** DUAL è una tabella costituita da una sola riga e da una sola colonna. Dato che la maggior parte delle funzioni di Oracle opera sia sulle colonne sia sui letterali, è possibile dimostrare alcune delle sue funzionalità usando semplicemente i letterali o le pseudocolonne. In tal caso, l'istruzione select non verifica quali colonne sono contenute nella tabella e una sola riga è più che sufficiente per la dimostrazione di un punto.

### ESEMPIO

L'esempio seguente mostra l'utente e la data di sistema correnti:

```
select Utente, SysDate from DUAL;
```

## DUMP

**VEDERE ANCHE** RAWTOHEX.

### SINTASSI

```
DUMP( stringa [.formato [.inizio [.lunghezza] ] ] )
```

**DESCRIZIONE** DUMP visualizza il valore di *stringa* nel formato interno, in ASCII, in notazione ottale, decimale, esadecimale o in formato carattere. *formato* vale per default ASCII o EBCDIC, a seconda della macchina; 8 produce la notazione ottale, 10 quella decimale, 16 quella esadecimale mentre 17 rappresenta la modalità carattere (analogia ad ASCII o EBCDIC). *inizio* rappresenta la posizione iniziale all'interno della stringa, mentre *lunghezza* è il numero di caratteri da visualizzare. *stringa* può essere un qualsiasi valore letterale o un'espressione.

**ESEMPIO**

L'esempio seguente mostra come rappresentare in formato esadecimale i caratteri dal primo all'ottavo per il valore Città della prima riga della tabella COMFORT:

```
select Citta, dump(Citta,16,1,8) from COMFORT where rownum < 2;
```

CITTA	DUMP(CITTA,16,1,8)
SAN FRANCISCO	Typ=1 Len= 13: 53,41,4e,20,46,52,41,4e

**EBCDIC**

EBCDIC è l'acronimo di Extended Binary Coded Decimal Interchange Code, la codifica utilizzata nei mainframe IBM e compatibili.

**ECHO (SQL\*PLUS)**

*Vedere SET.*

**EDIT**

**VEDERE ANCHE** DEFINE, SET, Capitolo 6.

**SINTASSI**

```
EDIT [file[.est]]
```

**DESCRIZIONE** EDIT chiama un editor di testo esterno standard e gli passa come argomento il nome del file. Se si omette *est*, viene aggiunta per default l'estensione 'SQL'. Se si omettono sia *file* sia *est*, viene chiamato l'editor e gli viene passato il nome di un file (inventato da SQLPLUS). Questo file include il contenuto del buffer corrente. La variabile utente locale \_EDITOR determina quale editor di testo debba essere utilizzato da EDIT. \_EDITOR può essere modificata mediante DEFINE, operazione che in genere viene eseguita al meglio all'interno del file LOGIN.SQL che viene letto ogni volta che si invoca SQL\*PLUS.

EDIT provoca la visualizzazione di un messaggio di errore se il buffer corrente è vuoto e se EDIT viene chiamato senza un nome di file. È possibile utilizzare il comando SET EDITFILE per impostare il nome predefinito del file creato dal comando EDIT.

**ELABORAZIONE DELLE TRANSAZIONI**

Si tratta di un tipo di elaborazione orientata all'uso di unità logiche di lavoro, piuttosto che a modifiche separate e individuali per mantenere la coerenza del database.

**ELABORAZIONE DISTRIBUITA**

L'elaborazione distribuita è una computazione effettuata su diverse CPU per ottenere un singolo risultato.

## **EMBEDDED (SQL\*PLUS)**

*Vedere SET.*

### **EMPTY\_BLOB**

**VEDERE ANCHE** Capitolo 32.

#### **SINTASSI**

`EMPTY_BLOB()`

**DESCRIZIONE** `EMPTY_BLOB` restituisce un locator LOB vuoto che può essere utilizzato per inizializzare una variabile LOB o, in un’istruzione `INSERT` o `UPDATE`, per inizializzare una colonna di tipo LOB o un attributo a `EMPTY`. `EMPTY` significa che il LOB è inizializzato ma non ancora popolato con i dati.

#### **ESEMPIO**

```
UPDATE TESTO_LIBRO set BlobCol = EMPTY_BLOB();
```

### **EMPTY\_CLOB**

**VEDERE ANCHE** Capitolo 32.

#### **SINTASSI**

`EMPTY_CLOB()`

**DESCRIZIONE** `EMPTY_CLOB` restituisce un locator LOB vuoto che può essere utilizzato per inizializzare una variabile LOB o, in un’istruzione `INSERT` o `UPDATE`, per inizializzare una colonna di tipo LOB o un attributo a `EMPTY`. `EMPTY` significa che il LOB è inizializzato ma non ancora popolato con i dati.

#### **ESEMPIO**

```
UPDATE TESTO_LIBRO set ClobCol = EMPTY_CLOB();
```

## **END**

**VEDERE ANCHE** BEGIN; BLOCCO, STRUTTURA; Capitolo 27.

#### **SINTASSI**

`END [etichetta blocco];`

**DESCRIZIONE** `END` non ha alcuna relazione con le istruzioni `END IF` ed `END LOOP`. Per ulteriori dettagli al riguardo, si *consulti* IF e LOOP.

`END` è l’istruzione di chiusura della sezione eseguibile di un blocco PL/SQL (e del blocco intero). Se nell’istruzione `BEGIN` è stato assegnato un nome al blocco, questo deve necessariamente seguire anche la parola `END`. Tra `BEGIN` ed `END` è richiesta la presenza di almeno un’istruzione eseguibile. Per ulteriori informazioni, si consulti la voce BLOCCO, STRUTTURA e i Capitoli 27 e 29.

## ENQUEUE

L'enqueue (accodamento) è un meccanismo di limitazione degli accessi di una particolare risorsa. Gli utenti o i processi in attesa della disponibilità della risorsa in questione non hanno ancora ottenuto l'enqueue. La maggior parte delle risorse contenute nei database consente l'utilizzo di questo meccanismo.

## ENTITÀ

Un'entità è una persona, un luogo o una cosa rappresentata da una tabella. In una tabella, ogni riga rappresenta un'occorrenza di questa entità.

## ENTITÀ-RELAZIONI, MODELLO

Un modello entità-relazioni è una struttura utilizzata nella realizzazione di modelli di sistemi per un database. Questo modello divide tutti gli elementi di un sistema in due categorie: le entità e le relazioni.

## EQUI-JOIN

Un equi-join è un join in cui l'operatore di confronto è un simbolo di uguaglianza. Per esempio:

```
where BIBLIOTECA.Titolo = BIBLIOTECA_PRESTITO.Titolo
```

## ESADECIMALE, NOTAZIONE

La notazione esadecimale è un sistema di numerazione in cui viene utilizzata la base 16 al posto della tradizionale base 10. I numeri tra 10 e 15 sono rappresentati con le lettere dalla A alla F. Questo sistema viene utilizzato spesso per visualizzare i dati interni memorizzati nei sistemi di elaborazione.

## ESCAPE (SQL\*PLUS)

*Vedere SET.*

## ESPRESSIONE

Un'espressione è un qualsiasi tipo di colonna. Il suo contenuto può essere un valore letterale, una variabile, un calcolo matematico, una funzione o una qualsiasi combinazione dei tipi appena enunciati; il risultato finale deve comunque essere un singolo valore, per esempio una stringa, un numero o una data.

## ESPRESSIONE LOGICA

Un'espressione logica è un'espressione che può assumere due soli valori: TRUE o FALSE. È un sinonimo di condizione.

## EXCEPTION

**VEDERE ANCHE** BLOCCO, STRUTTURA; DECLARE EXCEPTION; RAISE; Capitolo 27.

### SINTASSI

```
EXCEPTION
{WHEN {OTHERS | eccezione [OR eccezione]...}
    THEN istruzione; [istruzione;]...}
[ WHEN {OTHER | eccezione [OR eccezione]...}
    THEN istruzione; [istruzione;]...]...
```

**DESCRIZIONE** La sezione EXCEPTION di un blocco PL/SQL è quella cui passa il controllo del programma quando viene identificato un flag di eccezione. Questi possono essere definiti dall'utente o far parte delle eccezioni di sistema sollevate automaticamente da PL/SQL. Per ulteriori informazioni sui flag di eccezione definibili dall'utente, si consulti RAISE. I flag di eccezione del sistema sono tutti variabili BOOLEAN (che possono essere TRUE oppure FALSE). L'opzione WHEN viene impiegata per la loro valutazione. Per esempio, il flag NOT\_LOGGED\_ON viene sollevato quando si cerca di impartire un comando a Oracle senza aver effettuato l'accesso al sistema. Non è necessario dichiarare questo tipo di eccezione o sollevarne il flag. PL/SQL gestisce questa ed altre condizioni automaticamente.

All'atto dell'attivazione di un flag di eccezione, il programma si arresta immediatamente e il flusso dell'esecuzione passa alla sezione EXCEPTION del blocco corrente. Tuttavia, questa sezione non sa automaticamente quale flag di eccezione sia stato sollevato e non sa neanche cosa fare. Per questo motivo, occorre codificare prima la sezione EXCEPTION per controllare tutti i flag di eccezione che potrebbero essersi verificati, quindi impartire istruzioni per ciascuno. Questo è quanto viene svolto dal costrutto logico WHEN...THEN. È possibile utilizzare WHEN OTHERS come catch-all per gestire tutti gli errori inattesi per i quali non sono state dichiarate eccezioni.

### ESEMPIO

Nell'esempio che segue, viene valutato un flag di eccezione all'interno di una sezione EXCEPTION. Il flag ZERO\_DIVIDE viene sollevato quando si cerca di dividere per zero.

```
declare
    pi      constant NUMBER(9,7) := 3.1415927;
    raggio INTEGER(5);
    area    NUMBER(14,2);
    una_variabile  NUMBER(14,2);
begin
    raggio := 3;
    loop
        una_variabile := 1/(raggio-4);
        area := pi*power(raggio,2);
        insert into AREE values (raggio, area);
        raggio := raggio+1;
        exit when area >100;
    end loop;
exception
    when ZERO_DIVIDE
        then insert into AREE values (0,0);
end;
```

I flag di eccezione attivati dal sistema sono i seguenti:

---

ACCESS_INTO_NULL	Il programma cerca di assegnare dei valori agli attributi di un oggetto non inizializzato (automaticamente nullo).
CASE_NOT_FOUND	Nessuna delle opzioni nelle clausole WHEN di un'istruzione CASE è stata selezionata, e non esiste una clausola ELSE.
COLLECTION_IS_NULL	Il programma cerca di applicare metodi di collezione diversi da EXISTS a una tabella annidata o un array variabile non inizializzati (automaticamente nulli), oppure il programma cerca di assegnare dei valori agli elementi di una tabella annidata o di un array variabile non inizializzati.
CURSOR_ALREADY_OPEN	Il programma cerca di aprire un cursore già aperto. Prima di poter essere riaperto, un cursore deve essere stato chiuso. Un ciclo FOR a cursore apre automaticamente il cursore cui fa riferimento. Quindi, il programma non è in grado di aprire questo cursore all'interno del ciclo.
DUP_VAL_ON_INDEX	Il programma cerca di memorizzare valori duplicati nella colonna di un database che è vincolata da un indice unico.
INVALID_CURSOR	Il programma cerca di eseguire un'operazione di cursore non valida, come la chiusura di un cursore già chiuso.
INVALID_NUMBER	In un'istruzione SQL, la conversione di una stringa di caratteri in un numero non riesce perché la stringa non rappresenta un numero valido (nelle istruzioni procedurali, viene sollevata VALUE_ERROR). Questa eccezione viene sollevata anche quando l'espressione della clausola LIMIT in un'istruzione FETCH di tipo bulk non è valutata a un numero positivo.
LOGIN_DENIED	Il programma cerca di collegarsi a Oracle con un nome utente e/o una password non validi.
NO_DATA_FOUND	Un'istruzione SELECT INTO non restituisce righe, oppure il programma fa riferimento a un elemento cancellato in una tabella annidata o ad un elemento non inizializzato in una tabella a indice. Le funzioni di aggregazione SQL, come AVG e SUM, restituiscono sempre un valore oppure un valore NULL. Pertanto, un'istruzione SELECT INTO che chiama una funzione di aggregazione non solleva mai una NO_DATA_FOUND. Solitamente, l'istruzione FETCH non restituisce righe, pertanto, quando questa situazione si verifica, non viene sollevata alcuna eccezione.
NOT_LOGGED_ON	Il programma chiama il database senza essere collegato a Oracle.
PROGRAM_ERROR	PL/SQL ha un problema interno.
ROWTYPE_MISMATCH	La variabile host del cursore e la variabile del cursore PL/SQL coinvolte in un assegnamento hanno tipi di restituzione incompatibili. Per esempio, quando una variabile host del cursore aperta viene passata a un sottoprogramma memorizzato, i tipi di restituzione dei parametri effettivi e di quelli formali devono essere compatibili.
SELF_IS_NULL	Il programma cerca di chiamare un metodo MEMBER in un'istanza NULL. In altre parole, il parametro incorporato SELF (che è sempre il primo parametro passato a un metodo MEMBER) è NULL.
STORAGE_ERROR	PL/SQL ha esaurito la memoria oppure la memoria è stata danneggiata.
SUBSCRIPT_BEYOND_COUNT	Il programma fa riferimento a un elemento di una tabella annidata o di un array variabile con un numero di indice maggiore del numero di elementi nella collezione.
SUBSCRIPT_OUTSIDE_LIMIT	Il programma fa riferimento a un elemento di una tabella annidata o di un array variabile con un numero di indice (per esempio, - 1) che non rientra nell'intervallo valido.
SYS_INVALID_ROWID	La conversione di una stringa di caratteri in un valore di Rowid universale non riesce perché la stringa di caratteri non rappresenta un valore di Rowid valido.
TIMEOUT_ON_RESOURCE	Un timeout si verifica mentre Oracle attende una risorsa.
TOO_MANY_ROWS	Un'istruzione SELECT INTO restituisce più di una riga.
VALUE_ERROR	Si verifica un errore in un'operazione aritmetica, in una conversione, in un troncamento oppure di vincolo di dimensione. Per esempio, quando il programma seleziona il valore di una colonna in una variabile carattere, se il valore è più lungo della lunghezza dichiarata della variabile, PL/SQL interrompe l'assegnamento e solleva una VALUE_ERROR. Nelle istruzioni procedurali, viene sollevata una VALUE_ERROR se la conversione di una stringa di caratteri in un numero non riesce (nelle istruzioni SQL, viene sollevata INVALID_NUMBER).
ZERO_DIVIDE	Il programma cerca di dividere un numero per zero.

---

I numeri di errore e i valori del codice SQL impostati in Oracle sono i seguenti:

ECCEZIONE	# DI ERRORE ORACLE	CODICE SQL
ACCESS_INTO_NULL	ORA-06530	-6530
CASE_NOT_FOUND	ORA-06592	-6592
COLLECTION_IS_NULL	ORA-06531	-6531
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

**OTHERS** è l'opzione che permette di gestire tutti i flag di eccezione non valutati nella sezione **EXCEPTION**. Dev'essere sempre l'ultima istruzione WHEN a comparire e deve rimanere da sola. Non può essere inclusa in altre eccezioni.

La gestione delle eccezioni, specie quando si tratta di eccezioni dichiarate dall'utente, dovrebbe limitarsi agli errori irreversibili, ovvero quelli che provocano l'arresto della normale elaborazione.

Se viene attivato un flag di eccezione che non è stato controllato all'interno del blocco corrente, il programma salta alla sezione **EXCEPTION** del blocco via via più esterno, fino a che non trova l'eccezione sollevata, oppure il controllo torna al programma principale.

Le sezioni **EXCEPTION** possono riferirsi alle variabili nello stesso modo in cui lo fanno i blocchi dell'esecuzione. In altre parole, possono fare un riferimento diretto alle variabili locali o

alle variabili prese da altri blocchi facendole precedere dal nome del blocco in cui sono state definite.

Si presti particolare attenzione quando si controllano alcuni tipi di eccezioni. Per esempio, se la sezione EXCEPTION accetta il messaggio di errore e lo inserisce in una tabella del database, un'eccezione NOT\_LOGGED\_ON farebbe entrare la sezione EXCEPTION in un ciclo infinito. Si progettino le istruzioni che seguono THEN in modo da non rigenerare l'errore che si sta gestendo in un certo momento. È possibile utilizzare l'istruzione RAISE senza specificare alcun nome di eccezione. In questo modo, il controllo passa automaticamente alla sezione di gestione delle eccezioni del blocco più esterno, o al programma principale. Per ulteriori dettagli, si *consulti* RAISE.

## **EXCEPTION\_INIT**

**VEDERE ANCHE** DECLARE EXCEPTION, EXCEPTION, SQLCODE, guida di programmazione del precompilatore.

### **SINTASSI**

```
PRAGMA EXCEPTION_INIT(eccezione, intero);
```

**DESCRIZIONE** Le eccezioni di sistema standard, come ZERO\_DIVIDE, cui si fa riferimento per nome non sono altro che associazioni di nomi e numeri di errore interni di Oracle. Esistono centinaia di questi numeri di errore, ma solo ai più importanti sono stati assegnati dei nomi. Le eccezioni anonime attivano anch'esse dei flag e trasferiscono il controllo al blocco EXCEPTION, ma vengono gestite tutte da OTHERS, e in non in base al loro nome.

È possibile modificare questa impostazione assegnando nomi personalizzati agli altri numeri di errore di Oracle. L'istruzione EXCEPTION\_INIT consente di eseguire questa operazione. *eccezione* è il nome, composto da una sola parola, assegnato al numero di errore intero (si consulti il documento *Error Messages and Codes Manual* di Oracle per un elenco completo). *intero* dev'essere un numero negativo se lo è anche il codice di errore (ossia nel caso di errori irreversibili), mentre *eccezione* deve seguire le normali regole di denominazione degli oggetti.

Si osservi che la sintassi di questo comando richiede la presenza della parola PRAGMA prima di EXCEPTION\_INIT. Un pragma è una istruzione interna del compilatore PL/SQL, più che un vero e proprio codice eseguibile. Il pragma dev'essere posto nella sezione DECLARE di un blocco PL/SQL e dev'essere preceduto da una dichiarazione di eccezione.

### **ESEMPIO**

```
DECLARE
    qualche_spiacevole_errore      exception;
    pragma                          exception_init(qualche_spiacevole_errore, -660);
BEGIN
    ...
EXCEPTION
    when qualche_spiacevole_errore
        then ...
END;
```

## **EXECUTE**

**VEDERE ANCHE** CALL, CREATE PROCEDURE, CREATE FUNCTION, CREATE PACKAGE, CREATE PACKAGE BODY, GRANT, Capitoli 28 e 29.

## SINTASSI

```
EXECUTE nome Oggetto_procedurale [argomenti];
```

**DESCRIZIONE** EXECUTE esegue una procedura, un package o una funzione. Per eseguire un procedura all'interno di un package, è necessario specificare sia il nome del package sia quello della procedura sulla linea di comando, come risulta dall'esempio seguente. In questo esempio, viene eseguita una procedura di nome NUOVO\_LIBRO, in un package di nome PACKAGE\_BIBLIOTECA; il valore 'TALE OF TWO CITIES' viene passato come parametro di input alla procedura.

```
execute PACKAGE_BIBLIOTECA.NUOVO_LIBRO('TALE OF TWO CITIES');
```

Per eseguire un oggetto procedurale, si deve possedere il privilegio EXECUTE sull'oggetto in questione. Vedere GRANT.

## EXECUTE (Dynamic Embedded SQL)

**VEDERE ANCHE** EXECUTE IMMEDIATE, PREPARE, guida di programmazione del precompilatore.

### SINTASSI

```
EXEC SQL [FOR { intero | :dimensione_array }]
  EXECUTE id_istruzione
  [USING
    { DESCRIPTOR descrittore_SQLDA
    | :variabile_host [[INDICATOR] :variabile_indicatore]
    [, :variabile_host [[INDICATOR] :variabile_indicatore]]... } ]
```

**DESCRIZIONE** *intero* è una variabile host che serve per limitare il numero di iterazioni quando la clausola where utilizza degli array. *id\_istruzione* è l'identificativo per un'istruzione insert, delete o update pronta per essere eseguita (select non è ammessa). L'opzione USING introduce un elenco di sostituzioni a livello di variabili host delle istruzioni eseguite precedentemente.

## EXECUTE IMMEDIATE (Dynamic Embedded SQL)

**VEDERE ANCHE** EXECUTE, PREPARE, guida di programmazione del precompilatore.

### SINTASSI

```
EXEC SQL [AT {database | :variabile_host}]
  EXECUTE IMMEDIATE {:stringa | letterale}
```

**DESCRIZIONE** *database* è il nome di una connessione di database diversa da quella predefinita; *variabile\_host* può contenere un nome simile come proprio valore. *:stringa* è una stringa di variabili host contenente un'istruzione SQL. *letterale* è una stringa di caratteri contenente un'istruzione SQL. L'istruzione EXECUTE IMMEDIATE non può contenere riferimenti a variabili host diverse da un'istruzione SQL eseguibile in *:stringa*. L'istruzione SQL viene analizzata, resa eseguibile ed eseguita; al termine dell'esecuzione la forma eseguibile viene eliminata. Questa istruzione è utile solo nel caso di istruzioni che debbano essere eseguite una sola volta. Le istruzioni che richiedono reiterazioni dovrebbero utilizzare PREPARE ed EXECUTE eliminando così il lavoro ripetitivo e inutile di analisi di ogni istruzione.

## EXISTS

**VEDERE ANCHE** ANY, ALL, IN, NOT EXISTS, Capitolo 12.

### SINTASSI

```
select...  
      where EXISTS (select...)
```

**DESCRIZIONE** EXISTS restituisce *true* in una clausola where se la subquery che la segue restituisce almeno una riga. La clausola select nella subquery può essere una colonna, un letterale oppure un asterisco, non è importante (le convenzioni consigliano l'impiego di un asterisco o di una lettera 'x').

Molti programmatore preferiscono EXISTS a ANY e ALL in quanto risulta più semplice da ricordare e comprendere. Inoltre, la maggior parte delle formule gestibili con ANY e ALL può essere ricostruita con EXISTS.

### ESEMPIO

La query mostrata nell'esempio seguente si serve di EXISTS per elencare dalla tabella BIBLIOTECA tutti gli editori i cui libri sono stati presi in prestito:

```
select Editore  
      from BIBLIOTECA B  
     where EXISTS  
           (select 'x' from BIBLIOTECA_PRESTITO  
            where BIBLIOTECA_PRESTITO.Titolo = B.Titolo);
```

## EXISTSNODE

**VEDERE ANCHE** Capitolo 41.

### SINTASSI

```
EXISTSNODE (istanza_XMLType , stringa_Xpath )
```

**DESCRIZIONE** EXISTSNODE determina se l'analisi del documento eseguita con il percorso dà come risultato qualche nodo. Accetta come argomenti l'istanza XMLType che contiene un documento XML e una stringa VARCHAR2 che designa un percorso.

Il valore di restituzione è NUMBER: 0 se, dopo l'applicazione dell'analisi XPath al documento non rimane alcun nodo, >0 se rimane qualche nodo.

## EXIT (Forma 1, PL/SQL)

**VEDERE ANCHE** END, LOOP, Capitolo 27.

### SINTASSI

```
EXIT [ciclo] [WHEN condizione];
```

**DESCRIZIONE** Senza nessuna di queste opzioni, EXIT esce semplicemente dal ciclo in corso e sposta il controllo all'istruzione immediatamente successiva al ciclo. Se ci si trova in un ciclo annidato, è possibile uscire da qualsiasi livello del ciclo specificando il nome di quest'ultimo. Se si specifica una condizione, si potrà uscire dal ciclo quando la condizione risulta TRUE. Al-

l'uscita, qualsiasi cursore eventualmente aperto all'interno del ciclo viene chiuso automaticamente.

## **EXIT (Forma 2, SQL\*PLUS)**

**VEDERE ANCHE** COMMIT, DISCONNECT, QUIT, START.

### **SINTASSI**

```
{EXIT | QUIT} [SUCCESS|FAILURE|WARNING| intero|variabile]
[COMMIT ROLLBACK]
```

**DESCRIZIONE** EXIT termina una sessione SQL\*PLUS e riporta l'utente al sistema operativo, al programma chiamante o al menu principale. A meno che ROLLBACK non sia stato specificato, EXIT conclude qualsiasi operazione aperta sul database. Viene restituito anche un codice di ritorno. SUCCESS, FAILURE e WARNING hanno valori che sono specifici per ogni sistema operativo; FAILURE e WARNING possono essere uguali su alcuni sistemi operativi.

*intero* è un valore che si può restituire esplicitamente come codice di ritorno; *variabile* permette di impostare questo valore dinamicamente. Si può trattare di una variabile definita dall'utente oppure di una variabile di sistema come sql.sqlcode, che contiene sempre il codice SQL dell'ultima istruzione SQL eseguita in SQL\*PLUS o in un blocco PL/SQL incorporato.

## **EXP**

**VEDERE ANCHE** LN; NUMERI, FUNZIONI; Capitolo 8.

### **SINTASSI**

```
EXP(n)
```

**DESCRIZIONE** EXP restituisce il valore *e* elevato alla potenza di ordine *n*; *e* = 2.718281828...

## **EXPLAIN PLAN**

**VEDERE ANCHE** Capitolo 38.

### **SINTASSI**

```
EXPLAIN PLAN [SET STATEMENT_ID = 'nome']
[INTO [schema .] tabella [@dblink]]
FOR istruzione;
```

**DESCRIZIONE** *nome* identifica la spiegazione di questo piano per questa *istruzione* quando appare nella tabella di output, e segue le normali convenzioni di denominazione degli oggetti. Se *nome* non viene specificato, la colonna STATEMENT\_ID della tabella risulta NULL. *tabella* è il nome della tabella di output in cui devono comparire le informazioni relative al piano. La tabella dev'essere creata prima dell'esecuzione della procedura. Il file di avvio UTLXPLAN.SQL contiene il formato da utilizzare e può essere impiegato per creare la tabella predefinita (chiamata PLAN\_TABLE). Se non si specifica il parametro *tabella*, EXPLAIN PLAN utilizza PLAN\_TABLE. *istruzione\_sql* è il testo semplice di una qualsiasi istruzione select, insert, update o delete per la quale si desidera esaminare il piano di esecuzione di Oracle.

## EXPORT

Export può avere una di queste due definizioni:

- Export è l'utility di Oracle utilizzata per memorizzare dati del database Oracle in file con un formato di esportazione in modo che possano essere recuperati in un database Oracle mediante Import.
- L'esportazione è l'operazione che prevede l'uso dell'utility Export per scrivere i dati selezionati di una tabella in un file di esportazione.

Per informazioni sull'utility Export, si consulti la *Oracle9i Utilities User's Guide* e il Capitolo 40.

## EXTENT LIBERI

Gli extent liberi sono extent di blocchi del database non ancora allocati per alcuna tabella o segmento di indice. *Extent liberi* è quindi sinonimo di spazio libero.

## EXTENT USATI

I cosiddetti “extent usati” sono le zone di memoria che sono già state allocate a segmenti di dati (tabelle), pertanto possiedono già dei dati, oppure sono state riservate per i dati.

## EXTRACT (datetime)

**VEDERE ANCHE** DATE, FUNZIONI; Capitolo 9.

### SINTASSI

```
EXTRACT
( { { YEAR
      | MONTH
      | DAY
      | HOUR
      | MINUTE
      | SECOND      }
    | { TIMEZONE_HOUR
      | TIMEZONE_MINUTE      }
    | { TIMEZONE_REGION
      | TIMEZONE_ABBR
      }      }
  FROM { espressione_valore_dataora | espressione_valore_intervalllo } )
```

**DESCRIZIONE** EXTRACT estrae e restituisce il valore del campo di una data/ora specificata da una data/ora oppure da un'espressione del valore di intervallo.

### ESEMPIO

```
select Datanascita,
       EXTRACT(Mese from Datanascita) AS Mese
     from COMPLEANNO
   where Nome = 'VICTORIA';
```

DATANASCITA	MESE
-----	-----
20-MAY-49	5

## **EXTRACT (XML)**

**VEDERE ANCHE** Capitolo 41.

### **SINTASSI**

EXTRACT (*istanza\_XMLType*, *stringa\_Xpath* )

**DESCRIZIONE** EXTRACT assomiglia alla funzione EXISTSNODE. Si applica a una stringa XPath di tipo VARCHAR2 e restituisce un'istanza XMLType che contiene un frammento XML.

## **FEEDBACK (SQL\*PLUS)**

*Vedere SET.*

## **FETCH**

Una delle fasi dell'esecuzione di una query è quella di fetch: le righe di dati che soddisfano i criteri di ricerca vengono recuperati dal database.

### **FETCH (Forma 1, Embedded SQL)**

**VEDERE ANCHE** CLOSE, DECLARE CURSOR, DESCRIBE, VARIABILE INDICATORE, OPEN, PREPARE, guida di programmazione del precompilatore.

### **SINTASSI**

```
EXEC SQL [FOR { intero | :dimensione_array }]  
FETCH { cursoro | :variabile_cursoro }  
{ USING DESCRIPTOR descrittore_SQLDA  
| INTO :variabile_host [[INDICATOR] :variabile_indicatore]  
[, :variabile_host [[INDICATOR] :variabile_indicatore]]... }
```

**DESCRIZIONE** *intero* è una variabile host che imposta il numero massimo di righe da inserire nelle variabili di output. *cursoro* è il nome di un cursoro configurato precedentemente da DECLARE CURSOR. *:variabile\_host* corrisponde a una o più variabili host in cui vengono inseriti i dati recuperati tramite FETCH. Se anche solo una delle variabili host è un array, allora anche tutte le altre eventualmente presenti nell'elenco INTO devono essere array. La parola chiave INDICATOR è facoltativa.

### **FETCH (Forma 2, PL/SQL)**

**VEDERE ANCHE** %FOUND, %ROWTYPE, CLOSE, DECLARE CURSOR, LOOP, OPEN, SELECT . . . INTO, Capitolo 27.

## SINTASSI

```
FETCH cursore INTO {record | variabile [,variabile]...};
```

**DESCRIZIONE** FETCH recupera una riga di dati. L'istruzione di selezione relativa al cursore indicato determina le colonne da recuperare, mentre la relativa istruzione where determina quali e quante righe possono essere recuperate. Questo prende il nome di "blocco attivo", ma non può essere elaborato fino a che non lo si recupera riga per riga con l'istruzione FETCH. L'istruzione FETCH ottiene una riga dal blocco attivo e scarica i valori della riga all'interno del record (o nella stringa di variabili), che è stato definito in DECLARE.

Se si utilizza il metodo dell'elenco di variabili, a ciascuna colonna nel select list del cursore deve corrispondere una variabile e ognuna di esse dev'essere dichiarata nella sezione DECLARE. I tipi di dati devono essere uguali o al più compatibili.

Se invece si utilizza il metodo dei record, questi devono essere dichiarati con l'attributo %ROWTYPE secondo il quale la struttura del record è uguale (con i medesimi tipi di dati) a quella dell'elenco di colonne nella selezione. A ciascuna variabile del record si potrà accedere singolarmente usando come prefisso il nome del record, e il nome di una variabile identico al nome della colonna.

## ESEMPIO

```
declare
    pi      constant NUMBER(9,7) := 3.1415927;
    area    NUMBER(14,2);
    cursore rag_cursore is
        select * from VALORI_RAGGIO;
    val_rag rag_cursore%ROWTYPE;
begin
    open rag_cursore;
    loop
        fetch rag_cursore into val_rag;
        exit when rag_cursore%NOTFOUND;
        area := pi*power(val_rag.raggio,2);
        insert into AREE values (val_rag.raggio, area);
    end loop;
    close rag_cursore;
end;
```

## FIGLIO

Nei dati con struttura ad albero, viene chiamato figlio un nodo che discende immediatamente da un altro. Il nodo da cui discende l'elemento figlio viene chiamato *padre*.

## FILE, TIPI

Solitamente i file hanno un nome e un'estensione. Per esempio nel file comfort.tab, comfort rappresenta il nome mentre tab è l'estensione o "tipo" del file. In SQL\*PLUS, ai file di avvio creati con EDIT viene assegnata l'estensione predefinita SQL se non viene specificata alcuna estensione. In altre parole, il codice seguente:

```
edit misti
```

crea o modifica un file di nome misti.sql, e:

```
start misti
```

cerca di avviare un file di nome misti.sql dato che non è stata specificata alcuna estensione. Analogamente:

```
spool bellwood
```

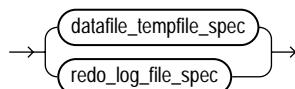
crea un file di output chiamato bellwood.lst, dato che LST è l'estensione predefinita dei file di spool. Naturalmente, quando edit e spool sono seguiti sia da un nome di file sia da un'estensione si ricorrerà all'estensione fornita e non a quella predefinita.

## **file\_specification**

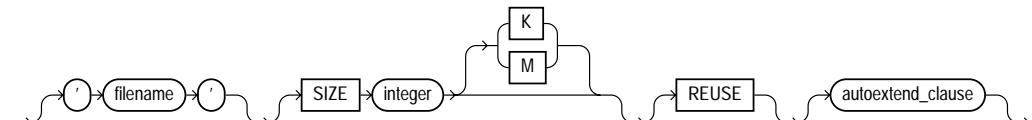
**VEDERE ANCHE** ALTER TABLESPACE, CREATE CONTROLFILE, CREATE DATABASE, CREATE LIBRARY, CREATE TABLESPACE.

### **SINTASSI**

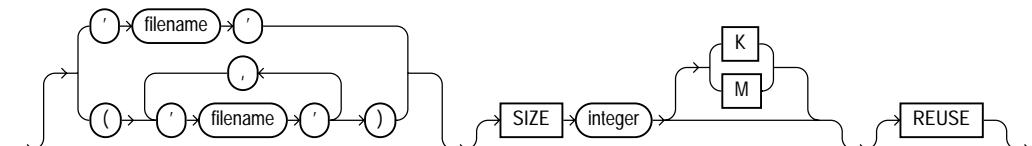
#### **file\_specification**



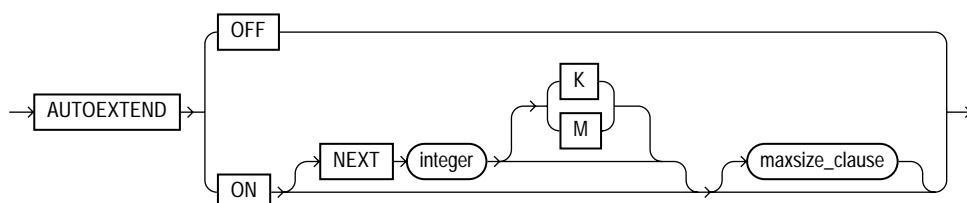
#### **datafile\_tempfile\_spec**

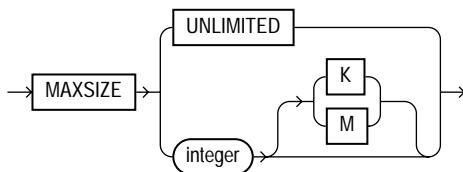


#### **redo\_log\_file\_spec**



#### **autoextend\_clause**



**maxsize\_clause**

**DESCRIZIONE** *file\_specification* definisce il nome, la locazione e la dimensione di un file utilizzato dal database. Per esempio, quando si crea una tablespace si deve specificare almeno un file di dati per la tablespace da utilizzare. *file\_specification* fornisce la sintassi per la parte del file di dati del comando CREATE TABLESPACE.

**FIRST**

**VEDERE ANCHE** DENSE\_RANK, LAST.

**SINTASSI**

```
funzione_aggregazione KEEP
( DENSE_RANK FIRST ORDER BY
  expr [ DESC | ASC ] [ NULLS { FIRST | LAST } ]
  [, expr [ DESC | ASC ] [ NULLS { FIRST | LAST } ] ]...
) [OVER clausola_partizionamento_query]
```

**DESCRIZIONE** FIRST è sia una funzione di aggregazione sia una funzione analitica che opera su un gruppo di valori presi da un insieme di righe che sono le prime a comparire rispetto a una determinata specifica di ordinamento. *funzione\_aggregazione* è una delle funzioni di gruppo numeriche standard (come MIN, MAX, AVG e STDDEV).

**FIRST\_VALUE**

**VEDERE ANCHE** LAST\_VALUE.

**SINTASSI**

```
FIRST_VALUE ( espr ) OVER (clausola_analitica)
```

**DESCRIZIONE** FIRST\_VALUE è una funzione analitica. Essa restituisce il primo valore in un gruppo ordinato di valori.

**FLOOR**

**VEDERE ANCHE** CEI; NUMERI FUNZIONI; Capitolo 8.

**SINTASSI**

```
FLOOR(valore)
```

**DESCRIZIONE** FLOOR è il valore intero più grande, minore o uguale al *valore* specificato.

**ESEMPIO**

```
FLOOR(2)      =  2
FLOOR(-2.3)  = -3
```

**FLUSH (SQL\*PLUS)**

*Vedere SET.*

**FOGLIA**

Nelle tabelle ad albero una foglia è una riga che non dispone di righe figlie.

**FOR**

*Vedere LOOP e CICLI A CURSORE.*

**FORMATO, MODELLO**

Un modello di formato è un'opzione che controlla l'aspetto di numeri, date e stringhe di caratteri. I modelli di formato per le colonne di tipo DATE sono utilizzati nelle funzioni di conversione della data, come TO\_CHAR e TO\_DATE.

**FROM**

**VEDERE ANCHE** DELETE, SELECT, Capitolo 3.

**SINTASSI**

```
DELETE FROM [utente.]tabella[@link] [alias]
WHERE condizione
SELECT... FROM [utente.]tabella[@link] [, [utente.]tabella[@link]
]...
```

**DESCRIZIONE** *tabella* è il nome della tabella utilizzata da delete o select. *link* è il collegamento a un database remoto. I comandi delete e select richiedono entrambi una clausola from per definire le tabelle da cui le righe devono essere selezionate o cancellate. Se la tabella è di proprietà di un altro utente, il nome della tabella deve essere preceduto da quello del proprietario.

Se la tabella è remota, si deve definire un database link. In tal caso, @link ne specifica il nome. Se si specifica link ma non utente, la query cerca una tabella di proprietà dell'utente sul database remoto. La *condizione* può includere una query correlata e utilizzare l'*alias* per la correlazione.

**FROM\_TZ**

**VEDERE ANCHE** TIMESTAMP WITH TIME ZONE, Capitolo 9.

**SINTASSI**

```
FROM_TZ ( valore_indicatore_orario, valore_fuso_orario )
```

**DESCRIZIONE** FROM\_TZ converte il valore di un indicatore orario a un certo fuso orario in un valore TIMESTAMP WITH TIME ZONE. *valore\_fuso\_orario* è una stringa di caratteri in formato ‘TZH:TZM’ o un’espressione di caratteri che restituisce una stringa in TZR con il formato TZD facoltativo.

## FULL OUTER JOIN

Un outer join completo esegue un outer join in cui le due tabelle vengono utilizzate come tabelle “esterne” nel join. Pertanto, A FULL OUTER JOIN B restituisce tutte le righe che A e B hanno in comune, tutte le righe prese da A che non hanno corrispondenze in B come valori NULL e tutte le righe di B che non hanno corrispondenze in A come valori NULL.

## FULL TABLE SCAN

Una scansione intera di tabella è un metodo per il recupero dei dati in cui Oracle cerca in modo sequenziale i dati specificati all’interno di tutti i blocchi del database per una tabella (piuttosto che ricorrere a un indice). *Vedere* il Capitolo 38.

## FUNZIONE

Una funzione è un’operazione predefinita, come “converti in maiuscole”, e può essere eseguita specificandone il nome e gli eventuali argomenti in un’istruzione SQL. *Vedere* CARATTERI, FUNZIONI; CONVERSIONE, FUNZIONI; DATA, FUNZIONI; FUNZIONI DI GRUPPO, FUNZIONI DI ELENCO, NUMERI, FUNZIONI e funzioni individuali.

## FUNZIONI DI ELENCO

**VEDERE ANCHE** Tutti gli altri elenchi di funzioni, Capitolo 9.

**DESCRIZIONE** Di seguito sono elencate, in ordine alfabetico, tutte le funzioni di elenco correnti di SQL di Oracle. Ciascuna viene riportata altrove in questa guida, assieme alla sintassi e ad esempi d’uso.

Questa funzione fornisce il primo valore non NULL del gruppo:

COALESCE(*valore1*, *valore2*, ...)

Questa funzione fornisce il valore massimo di un elenco:

GREATEST(*valore1*, *valore2*, ...)

Questa il valore minimo:

LEAST(*valore1*, *valore2*, ...)

## FUNZIONI DI GRUPPO

**VEDERE ANCHE** AVG; COUNT; MAX; MIN; NUMERI, FUNZIONI; STDDEV; SUM; VARIANZA; Capitolo 8.

**DESCRIZIONE** Una funzione di gruppo calcola un singolo valore riassuntivo (come una somma o una media) dai singoli valori numerici in un gruppo di valori. Questo è un elenco alfabetico di tutte le funzioni di gruppo correnti in SQL di Oracle. Ciascuna viene riportata altrove in questa guida, assieme alla sintassi e ad esempi d'uso. Le funzioni di gruppo risultano utili solo all'interno di query e subquery. DISTINCT fa in modo che una funzione di gruppo riassume solo valori distinti (unici).

## NOME E IMPIEGO DELLE FUNZIONI

AVG fornisce la media dei valori contenuti in un gruppo di righe.

CORR restituisce il coefficiente di correlazione di un gruppo di coppie di numeri.

COUNT fornisce il numero di righe di una colonna o di una tabella (quando si usa \*).

COVAR\_POP restituisce la covarianza della popolazione di un gruppo di coppie di numeri.

COVAR\_SAMP restituisce la covarianza del campione di un gruppo di coppie di numeri.

CUME\_DIST calcola la distribuzione cumulativa di un valore in un gruppo di valori.

DENSE\_RANK calcola la posizione di una riga in un gruppo ordinato di righe.

FIRST opera su un insieme di valori presi da un gruppo di righe che sono le prime a comparire rispetto a una determinata specifica di ordinamento.

FIRST\_VALUE restituisce il primo valore in un gruppo ordinato di valori.

GROUP\_ID distingue i gruppi duplicati che risultano da una specifica GROUP BY.

GROUPING distingue le righe di grossi aggregati da righe raggruppate regolarmente.

GROUPING\_ID restituisce un numero che corrisponde al vettore bit di GROUPING associato a una riga.

LAST opera su un insieme di valori presi da un gruppo di righe che compaiono per ultime rispetto a una determinata specifica di ordinamento.

MAX fornisce il massimo tra tutti i valori di un gruppo di righe.

MIN fornisce il minimo tra tutti i valori di un gruppo di righe.

PERCENT\_RANK esegue un calcolo per stabilire la classificazione percentuale.

PERCENTILE\_CONT è una funzione di calcolo percentile che assume un modello di distribuzione continuo.

PERCENTILE\_DISC è una funzione di calcolo percentile che assume un modello di distribuzione discreta.

RANK calcola la posizione di un valore in un gruppo di valori.

Le funzioni REGR eseguono analisi di regressione lineari su un gruppo di valori.

STDDEV fornisce la deviazione standard di tutti i valori di un gruppo di righe.

STDDEV\_POP calcola la deviazione standard della popolazione e restituisce la radice quadrata della varianza della popolazione.

STDDEV\_SAMP calcola la deviazione standard del campione cumulativo e restituisce la radice quadrata della varianza del campione.

SUM fornisce la somma di tutti i valori di un gruppo di righe.

VAR\_POP restituisce la varianza della popolazione di un insieme di numeri dopo aver scartato i valori NULL contenuti in questo gruppo.

VAR\_SAMP restituisce la varianza del campione di un insieme di numeri dopo aver scartato i valori NULL contenuti in questo gruppo.

VARIANCE fornisce la varianza di tutti i valori di un gruppo di righe.

## FUNZIONI DI REGRESSIONE

A partire da Oracle9i, Oracle supporta le seguenti funzioni di regressione: REGR\_SLOPE, REGR\_INTERCEPT, REGR\_COUNT, REGR\_R2, REGR\_AVGX, REGR\_AVGY, REGR\_SXX, REGR\_SYY e REGR\_SXY. Per la sintassi ed esempi di queste funzioni analitiche, si consulti la *Oracle SQL Reference Guide*.

## GET

**VEDERE ANCHE** EDIT, SAVE.

### SINTASSI

GET *file* [LIST | NOLIST]

**DESCRIZIONE** GET carica il file di un sistema host, di nome *file*, nel buffer corrente (ossia quello di SQL o uno definito dall'utente). Se il tipo di file non viene specificato, GET suppone si tratti di un file SQL. LIST fa in modo che SQL\*PLUS elenchi le linee che sono state caricate nel buffer. Questa è l'impostazione predefinita. NOLIST carica il file senza elencarne il contenuto.

### ESEMPIO

L'esempio seguente ottiene un file di nome lavoro.sql:

```
get lavoro
```

## GOTO

**VEDERE ANCHE** BLOCCO, STRUTTURA; Capitolo 27.

### SINTASSI

GOTO *etichetta*;

**DESCRIZIONE** GOTO trasferisce il controllo alla sezione di codice preceduta dall'*etichetta*. Questa può precedere qualsiasi istruzione valida all'interno di un blocco dell'esecuzione o in una sezione EXCEPTION. Tuttavia, è necessario tener conto di alcune limitazioni:

- GOTO non può trasferire il controllo a un blocco racchiuso nel blocco corrente, o contenuto in un ciclo FOR o un'istruzione IF.
- GOTO non può trasferire il controllo a etichette esterne al proprio blocco a meno che non si tratti del blocco immediatamente esterno.

### ESEMPIO

Ecco un ciclo informale costruito a partire da GOTO che, senza conoscere a priori il numero massimo, inserisce in una tabella una progressione geometrica dei numeri da 1 a (circa) 10.000:

```
DECLARE
  ...
BEGIN
  x := 0;
  y := 1;
  <<alpha>>
  x := x + 1;
  y := x*y;
```

```

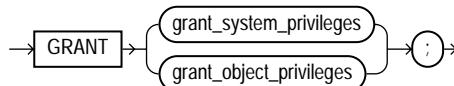
insert ...
if y > 10000
  then goto beta;
  goto alpha;
<<beta>>
  exit;

```

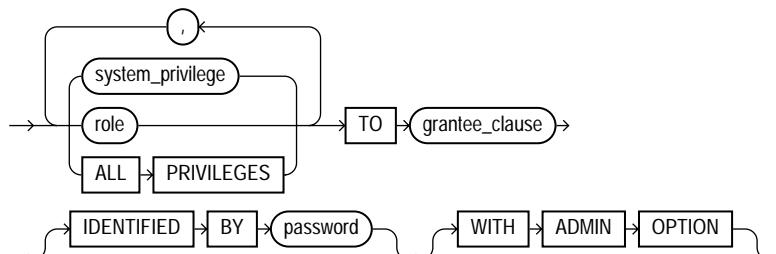
## GRANT

**VEDERE ANCHE** ALTER USER, CREATE USER, PRIVILEGI, REVOKE, RUOLI, Capitolo 19.  
**SINTASSI**

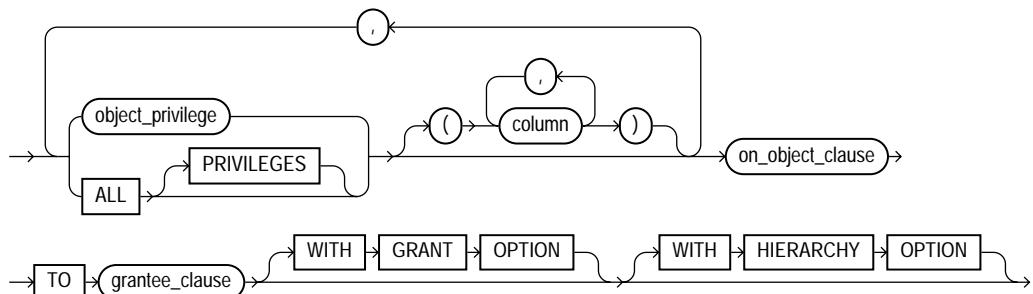
grant



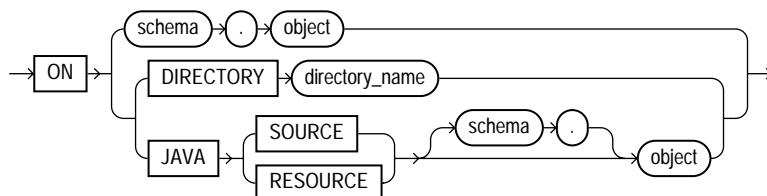
**grant\_system\_privileges**

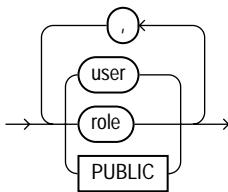


**grant\_object\_privileges**



**on\_object\_clause**



**grantee\_clause**

**DESCRIZIONE** La prima forma di GRANT estende uno o più privilegi di sistema a utenti e ruoli. Un privilegio di sistema non è altro che un'autorizzazione all'esecuzione di particolari comandi per il controllo o la definizione dei dati, come ALTER SYSTEM, CREATE ROLE o GRANT. Per ulteriori informazioni sui privilegi, si consulti PRIVILEGI. Gli utenti vengono creati con CREATE USER, in seguito vengono concessi i privilegi di sistema che permettono loro la connessione al database e l'esecuzione di alcune operazioni. Un utente privo di concessioni non è in grado di eseguire alcuna operazione in Oracle. Un *ruolo* è una collezione di privilegi creata con il comando CREATE ROLE. Come accade per gli utenti, anche un ruolo quando viene creato non dispone di alcun privilegio. A entrambe le entità i privilegi vengono assegnati per mezzo di uno o più GRANT. Inoltre, è possibile concedere ruoli ad altri ruoli per creare una rete annidata di permessi, che offre all'amministratore del database un'ampia libertà nella gestione della sicurezza del database.

L'opzione WITH ADMIN OPTION consente all'utente o al ruolo (il concessionario), di concedere il sistema di privilegio o il ruolo ad altri concessionari. Il concessionario può anche modificare o cancellare l'opzione WITH ADMIN OPTION concessa a un ruolo.

La seconda forma del comando GRANT concede privilegi relativi agli oggetti, ovvero privilegi che riguardano tipi specifici di oggetti nei database Oracle, di qualunque utente o ruolo. *oggetto* può essere una tabella, vista, sequenza, procedura, funzione, package, vista materializzata o un sinonimo per uno di questi oggetti. Le concessioni di privilegi a un sinonimo equivalgono alle concessioni dei medesimi privilegi agli oggetti di base cui fa riferimento il sinonimo. Per un elenco completo dei privilegi degli oggetti e del loro significato, si consulti la parte relativa ai PRIVILEGI.

Se si concedono privilegi INSERT, REFERENCES o UPDATE su una tabella o vista, il comando GRANT può anche specificare un elenco di colonne della tabella o della vista cui applicare i permessi. Il comando GRANT verrà applicato soltanto a queste colonne. Se GRANT non specifica un elenco di colonne, i permessi vengono applicati a tutte le colonne della tabella o della vista.

PUBLIC concede i privilegi a tutti gli utenti attuali e futuri.

L'opzione WITH GRANT OPTION concede agli utenti (o ai ruoli) di destinazione il permesso di concedere il medesimo privilegio ad altri utenti o ruoli.

## GREATEST

**VEDERE ANCHE** ORDINAMENTO, LEAST, MAX, FUNZIONI DI ELENCO, Capitoli 8 e 9.  
**SINTASSI**

GREATEST(*valore1*, *valore2*, ...)

**DESCRIZIONE** GREATEST sceglie il numero più grande da un elenco di valori. Questi possono essere colonne, valori letterali o espressioni, e tipi di dati CHAR, VARCHAR2, NUMBER e

**DATE.** Un numero con un valore più grande è considerato “maggiore” di quello con il valore più piccolo. Tutti i numeri negativi vengono considerati minori di quelli positivi. Perciò –10 è minore di 10; –100 è minore di –10.

Una data successiva è considerata maggiore di una precedente.

Le stringhe di caratteri vengono confrontate lettera per lettera a partire da quella più a sinistra fino al primo carattere differente. La stringa che ha il carattere più grande in quella posizione è considerata la “maggiore”. Un carattere è considerato più grande di un altro quando ha un codice ASCII (o EBCDIC) più elevato. Questo significa generalmente che B è maggiore di A ma anche che il valore di A in rapporto ad “a” o a “1” varia da piattaforma a piattaforma.

Se due stringhe sono identiche fino alla fine di quella più piccola, la stringa più lunga è considerata maggiore. Se due stringhe sono identiche e per di più hanno la stessa lunghezza sono considerate uguali. In SQL, è essenziale che tutti i letterali numerici vengano inseriti senza apici: ‘10’ verrebbe considerato minore di ‘6’ poiché gli apici indicano a SQL di considerarli come stringhe di caratteri; il ‘6’ è più grande della cifra 1 di ‘10’.

A differenza della maggior parte delle funzioni e degli operatori logici di Oracle, GREATEST e LEAST non considerano come date le stringhe di caratteri eventualmente corrispondenti a tale formato. Per fare in modo che LEAST e GREATEST funzionino correttamente, si deve applicare per prima la funzione TO\_DATE ai letterali da valutare.

## GROUP BY

**VEDERE ANCHE** CUBE, HAVING, ORDER BY, ROLLUP, WHERE, Capitolo 11.

### SINTASSI

```
SELECT espressione [.espressione]...
  GROUP BY espressione [.espressione]...
    HAVING condizione
      ...
```

**DESCRIZIONE** GROUP BY fa in modo che una clausola select produca una riga di riepilogo per tutte le righe selezionate che hanno valori identici in una o più colonne o espressioni specificate dall’utente. Ogni espressione nella clausola select deve essere una di queste:

- una costante;
- una funzione senza parametri (SysDate, User);
- una funzione di gruppo come SUM, AVG, MIN, MAX, COUNT;
- un’espressione identica a quelle specificate nella clausola GROUP BY.

Le colonne specificate nella clausola GROUP BY possono anche non essere ripetute nella clausola select; mentre devono essere presenti all’interno della tabella.

Per determinare quali gruppi includere nella clausola GROUP BY si deve utilizzare HAVING. Una clausola where, invece, stabilisce quali righe includere nei gruppi.

GROUP BY e HAVING seguono WHERE, CONNECT BY e START WITH. La clausola ORDER BY viene eseguita dopo le clausole WHERE, GROUP BY e HAVING (le quali a loro volta vengono eseguite in questo ordine). Inoltre, può implementare funzioni di gruppo o colonne tratte da GROUP BY, oppure una combinazione di entrambe. Se utilizza una funzione di gruppo, questa funzione agisce sui gruppi, poi ORDER BY ordina i *risultati* della funzione. Se ORDER BY impiega una colonna di GROUP BY, ordina le righe restituite in base a questa colonna. Le funzioni di gruppo e le singole colonne possono essere combinate in ORDER BY (purché la colonna si trovi in GROUP BY).

Nella clausola ORDER BY è possibile specificare una funzione di gruppo e la colonna su cui agisce, anche se queste non hanno nulla a che vedere con le funzioni di gruppo e le colonne delle clausole SELECT, GROUP BY o HAVING. D'altra parte, se si specifica una colonna in ORDER BY che non appartiene a una funzione di gruppo, questa colonna deve essere contenuta nella clausola GROUP BY.

### ESEMPIO

```
select Titolo, COUNT(*)  
  from BIBLIOTECA_AUTORE  
 group by Titolo  
having COUNT(*) = 1;
```

## GROUP\_ID

**VEDERE ANCHE** GROUP BY, GROUPING\_ID, FUNZIONI DI GRUPPO, Capitolo 13.

### SINTASSI

GROUP\_ID ( )

**DESCRIZIONE** GROUP\_ID distingue i gruppi duplicati che vengono prodotti da una specifica GROUP BY.

## GROUPING

GROUPING è una funzione utilizzata insieme alle funzioni ROLLUP e CUBE per rilevare i valori NULL. *Vedere* ROLLUP.

## GROUPING\_ID

GROUPING\_ID restituisce un numero che corrisponde al vettore bit di GROUPING associato a una riga. Si consultino ROLLUP, CUBE, e il Capitolo 13.

## GRUPPO DI AGGIORNAMENTO DI SNAPSHOT

Un gruppo di aggiornamento di snapshot è un insieme di viste materializzate locali il cui aggiornamento avviene contemporaneamente. I gruppi di aggiornamento permettono di imporre la coerenza dei dati tra le viste materializzate. *Vedere* il Capitolo 23.

## HASH JOIN

Un hash join è un metodo che permette di unire tramite join due tabelle. Questo metodo utilizza algoritmi di hash (simili a quelli utilizzati per i cluster di hash) per valutare quali righe soddisfano i criteri di join. Sebbene utilizzino algoritmi simili, non esiste alcuna relazione diretta tra gli hash join e i cluster di hash.

## HAVING

Vedere GROUP BY.

## HEADING (SQL\*PLUS)

Vedere SET.

## HEADSEP (SQL\*PLUS)

Vedere SET.

## HELP (SQL\*PLUS)

### SINTASSI

HELP [*argomento*]

**DESCRIZIONE** Il comando HELP accede al sistema di guida di SQL\*PLUS. HELP INDEX visualizza un elenco di argomenti.

## HEXTORAW

**VEDERE ANCHE** RAWTOHEX.

### SINTASSI

HEXTORAW(*stringa\_esadecimale*)

**DESCRIZIONE** HEXTORAW trasforma una stringa di caratteri di numeri esadecimali in notazione binaria.

## HOST (SQL\*PLUS)

**VEDERE ANCHE** \$, @, @@, START.

### SINTASSI

HO[ST] *host comando*

**DESCRIZIONE** Un host è un computer su cui viene eseguito Oracle RDBMS. Il comando HOST in SQL\*PLUS restituisce qualsiasi comando dell'host al sistema operativo in modo che possa essere eseguito senza uscire da SQL\*PLUS. SQL\*PLUS permette di incorporare variabili locali o PL/SQL nella stringa di comando dell'host. Questa funzione purtroppo non garantisce il funzionamento su tutte le possibili piattaforme e sistemi operativi.

## IF

**VEDERE ANCHE** CICLI, Capitolo 27.

### SINTASSI

```
[IF condizione THEN istruzione; [istruzione;]...  
[ELSIF condizione THEN istruzione; [istruzione;]...  
[ELSIF condizione THEN istruzione; [istruzione;]... ]...  
[ELSE istruzione; [istruzione;]...] ]  
END IF;
```

**DESCRIZIONE** L'istruzione IF esegue una o più *istruzioni* se *condizione* vale TRUE, quindi il programma salta alla riga che segue END IF. Se *condizione* vale FALSE, vengono valutate tutte le condizioni ELSIF facoltative. Se una vale TRUE, vengono eseguite le *istruzioni* associate (quelle che seguono THEN) e quindi il programma salta a END IF. Se nessuna delle condizioni ELSIF è TRUE (e la condizione IF originale non era TRUE), vengono eseguite le istruzioni che seguono l'istruzione ELSE facoltativa. Si noti che ELSE finale non ha alcuna condizione.

### ESEMPIO

```
declare  
    pi      constant NUMBER(9,7) := 3.1415927;  
    area   NUMBER(14,2);  
    cursole rag_cursore is  
        select * from VALORI_RAGGIO;  
    val_rag rag_cursore%ROWTYPE;  
begin  
    open rag_cursore;  
    fetch rag_cursore into val_rag;  
    area := pi*power(val_rag.raggio,2);  
    if area >30  
    then  
        insert into AREE values (val_rag.raggio, area);  
    end if;  
    close rag_cursore;  
end;
```

## IMPORT

Import è l'utility di Oracle che serve per recuperare i dati del database Oracle trovati in file con formato di esportazione contenuti in un database Oracle. L'importazione è un'operazione che prevede l'impiego dell'utility Import per spostare i dati da un file di esportazione nelle tabelle del database. Vedere EXPORT. Per i dettagli sull'impiego di Import, si consulti la *Oracle9i Utilities Guide* e il Capitolo 40.

## IN

**VEDERE ANCHE** ALL, ANY, OPERATORI LOGICI, WHERE, Capitolo 3.

### SINTASSI

```
WHERE espressione IN ({'stringa' [, 'stringa']... | select...})
```

**DESCRIZIONE** IN equivale a =ANY. Nella prima opzione, IN significa che l'espressione è uguale a qualsiasi membro dell'elenco seguente di *stringhe* letterali. Nella seconda opzione, significa che l'espressione è uguale a qualsiasi valore in qualsiasi riga selezionata dalla subquery. Si tratta di due costrutti logicamente equivalenti, con il primo che fornisce un elenco composto da stringhe letterali e il secondo che crea un elenco da una query. IN funziona con i tipi di dati VARCHAR2, CHAR, DATE e NUMBER, e anche con i tipi RowID.

## INDICE

Indice è un termine generale che identifica una caratteristica di Oracle/SQL utilizzata principalmente per velocizzare l'esecuzione e imporre l'univocità di alcuni dati. In generale, gli indici offrono un metodo di accesso più rapido ai dati di una tabella rispetto all'esecuzione di una scansione intera di tabella. Esistono diversi tipi di indici; vedere INDICE CONCATENATO, INDICE COMPRESSO e INDICE UNICO. Un indice ha una voce per ciascun valore che si trova nei campi indicizzati della tabella (eccetto quelli con un valore NULL) e puntatori alle righe che hanno questo valore.

## INDICE COMPRESSO

Un indice compresso è un indice per il quale vengono memorizzate solo le informazioni necessarie a identificare le voci non duplicate; le informazioni che un indice memorizza con la chiave precedente o successiva sono "compresse" (truncate) e non registrate per ridurre lo spazio richiesto dall'indice. Vedere anche INDICE NON COMPRESSO.

## INDICE CONCATENATO (o CHIAVE)

Un indice concatenato è un indice creato su più colonne di una tabella. Può essere utilizzato per garantire che quelle colonne abbiano valori unici per ciascuna riga della tabella e per velocizzare l'accesso alle righe per mezzo di tali colonne. Vedere CHIAVE COMPOSTA.

## INDICE GLOBALE

Quando si partiziona una tabella, i suoi dati vengono memorizzati in segmenti separati. Quando sulla tabella partizionata viene creato un indice, questo può essere partizionato a sua volta per rispecchiare ogni partizione della tabella. Se le partizioni degli indici non coincidono con quelle delle tabelle, l'indice si dice *globale*. Vedere il Capitolo 18.

## INDICE LOCALE

Quando si partiziona una tabella, i suoi dati vengono memorizzati all'interno di tabelle diverse. Quando su una tabella partizionata si crea un indice, questo può essere partizionato in modo da rispecchiare ogni partizione della tabella. Queste partizioni indicizzate prendono il nome di indici locali. Vedere il Capitolo 18.

## INDICE REVERSE

In un indice Reverse, i byte del valore indicizzato vengono invertiti prima di essere memorizzati nell'indice stesso. Pertanto, valori come 1234 e 1235 vengono indicizzati come se fossero 4321 e 5321. Come conseguenza di questa inversione, i due valori vengono memorizzati in locazioni ancora più distanti. Se si esegue spesso una query sulla colonna per trovare una corrispondenza esatta (*colonna=1234*) un indice Reverse potrebbe ridurre i conflitti di blocco all'interno dell'indice. Se si eseguono spesso query di intervallo (*colonna1234* o *colonna like '123%*) allora gli indici tradizionali sono più adatti alle proprie esigenze. Vedere CREATE INDEX.

## INDICE TESTUALE

Un indice testuale è un insieme di tabelle e indici utilizzati dai programmi per le ricerche di testo. Come l'indice di un libro, un indice testuale contiene le tabelle con le voci di testo e le relative posizioni. Vedere il Capitolo 24.

## INDICE UNICO

Un indice unico è un indice che impone l'univocità di ogni valore che indicizza. L'indice può essere un'unica colonna o un tipo di dati concatenato (formato di diverse colonne). Vedere VINCULO DI INTEGRITÀ.

## INDICI, SEGMENTO

Il segmento di un indice è la memoria allocata per un indice, in contrapposizione alla memoria allocata per i dati di una tabella.

## init.ora

init.ora è un file dei parametri di sistema del database che contiene impostazioni e nomi di file utilizzati all'avvio di un sistema mediante il comando CREATE DATABASE o i comandi di avvio e di uscita. In Oracle9i è possibile utilizzare un file init.ora oppure si può creare un file dei parametri di sistema mediante il comando CREATE SPFILE. Vedere CREATE PFILE e CREATE SPFILE.

## INITCAP

**VEDERE ANCHE** CARATTERI, FUNZIONI; LOWER; UPPER; Capitolo 7.

### SINTASSI

INITCAP(*stringa*)

**DESCRIZIONE** INITCAP rende maiuscola la prima lettera di una o più parole. Inoltre, tiene conto della presenza di simboli e opera le modifiche sulle lettere che seguono spazi e simboli come virgole, punti, due punti, punto e virgola, !, @, #, \$ e così via.

## ESEMPIO

`INITCAP('ecco.qui.un-esempio di !come@initcap#funziona')`

produce:

Ecco.Qui.Un-Esempio Di!Come@Initcap#Funziona

## INPUT

**VEDERE ANCHE** APPEND, CHANGE, DEL, EDIT, Capitolo 6.

### SINTASSI

`I[INPUT] [testo]`

**DESCRIZIONE** INPUT aggiunge una nuova linea di testo dopo la linea corrente nel buffer corrente. INPUT senza parametri permette di inserire più linee dopo la linea corrente, che si interrompe quando il sistema incontra un carattere di INVIO su una linea vuota. Lo spazio tra INPUT e *testo* non viene aggiunto alla linea, mentre qualsiasi altro spazio viene aggiunto. Per una discussione relativa alla linea corrente, si *consulti* DEL.

## INSERT (Forma 1, Embedded SQL)

**VEDERE ANCHE** EXECUTE, FOR, guida di programmazione del precompilatore.

### SINTASSI

```
EXEC SQL [AT { db_nome | :variabile_host } ]
[FOR { :intero_host | :intero }]
INSERT INTO
[ (subquery)
| [schema.] { tabella | vista }
[ @db_link | PARTITION (nome_part) ] ]
[(colonna) [, (colonna)]...]
{ VALUES (espr [.espr]...) | subquery }
{|{ RETURN | RETURNING } espr [. espr]... INTO
:variabile_host [[INDICATOR] :variabile_ind]
[, :variabile_host [[INDICATOR] :variabile_ind]]...]
```

**DESCRIZIONE** *db\_nome* è un database diverso da quello predefinito, mentre *variabile\_host* contiene il nome del database. *:intero\_host* è un valore host che limita il numero di iterazioni di INSERT (*vedere FOR*). *tabella* è il nome di una tabella, di una vista o di un sinonimo mentre *db\_link* è il nome di un database remoto in cui è stata memorizzata la tabella (per una discussione sulla clausola VALUES, sull'uso delle colonne e di query si consulti la definizione di INSERT, Forma 3). *espressione* può essere un'espressione vera e propria o una variabile host nella forma *:variabile[:indicatore]*.

## INSERT (Forma 2, PL/SQL)

**VEDERE ANCHE** SQL CURSOR, Capitolo 27.

## SINTASSI

```
INSERT INTO [utente.]tabella[@db_link] [(colonna [,colonna]...)]
VALUES (espressione_[,espressione]...) | query...;
```

**DESCRIZIONE** La modalità di impiego di INSERT in PL/SQL è identica alla sua forma generale (Forma 3 del paragrafo seguente) salvo le seguenti eccezioni.

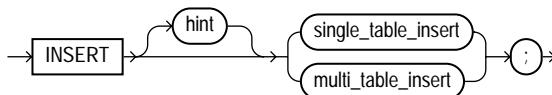
- È possibile utilizzare una variabile PL/SQL in un'espressione per il valore dell'elenco VALUES.
- Ogni variabile viene trattata allo stesso modo di una costante contenente il valore della variabile.
- Se si usa la versione *query...* di INSERT, non si può utilizzare la clausola INTO di select.

## INSERT (Forma 3, comando SQL)

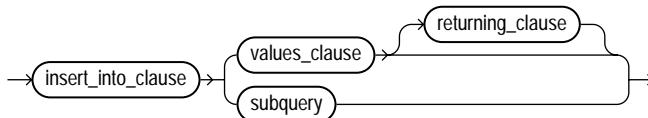
**VEDERE ANCHE** CREATE TABLE, Capitoli 4, 15, 30 e 31.

### SINTASSI

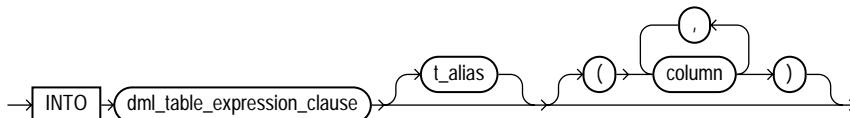
insert



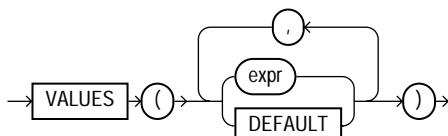
single\_table\_insert



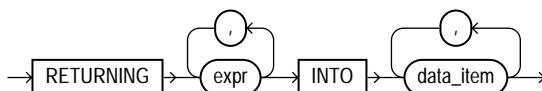
insert\_into\_clause

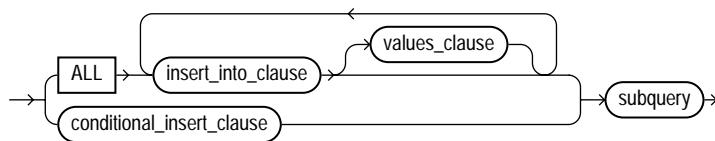
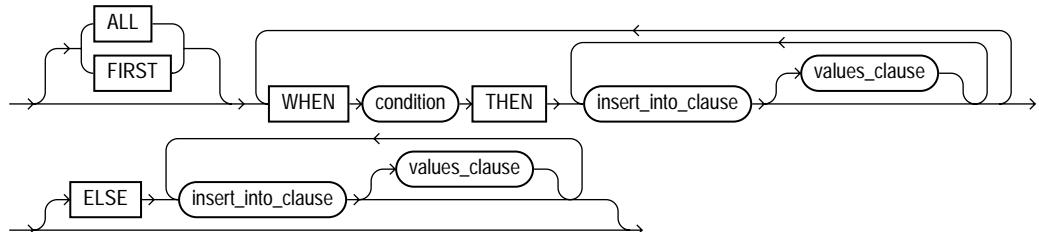
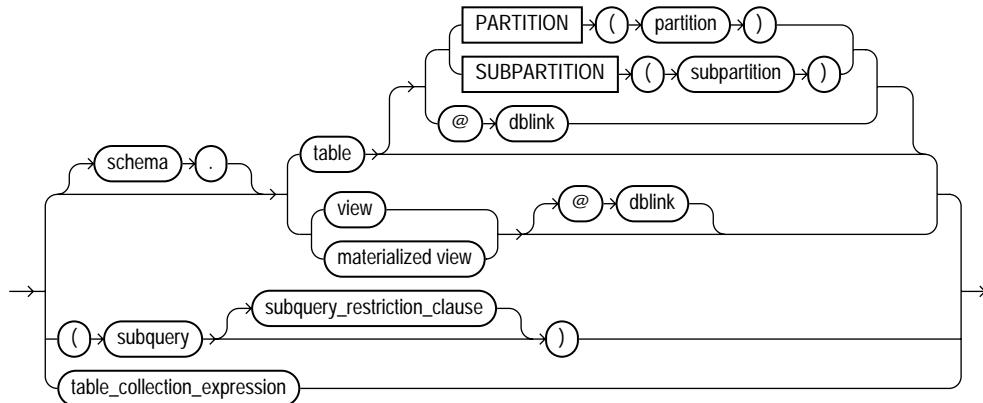
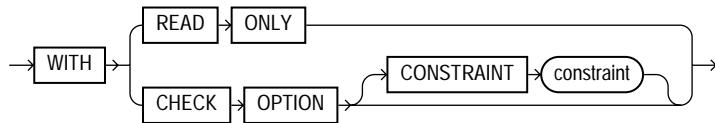


values\_clause



returning\_clause



**multi\_table\_insert****conditional\_insert\_clause****DML\_table\_expression\_clause****subquery\_restriction\_clause****table\_collection\_expression**

**DESCRIZIONE** `INSERT` aggiunge una o più righe nuove alla tabella o alla vista. Il campo facoltativo `schema` dev'essere un utente per la cui tabella è stato concesso il privilegio di inserimento. `tabella` è la tabella in cui inserire le righe. Se viene fornito un elenco di colonne, per ciascuna dev'essere verificata un'espressione SQL. Le colonne non presenti in elenco ricevono

il valore NULL e nessuna di esse deve essere stata definita NOT NULL altrimenti INSERT fallisce. Se non viene fornito un elenco di colonne, per tutte le colonne della tabella si dovranno specificare dei valori.

INSERT con una subquery aggiunge il numero di righe risultanti dalla query associando, posizione per posizione, ogni colonna della query alle colonne contenute nell'elenco. Se non viene fornito alcun elenco di colonne, le tabelle dovranno avere lo stesso numero e tipo di colonne. La vista del dizionario dati USER\_TAB\_COLUMNS fornisce un esempio, così come DESCRIBE in SQL\*PLUS.

### ESEMPIO

Di seguito, viene inserita una riga nella tabella COMFORT:

```
insert into COMFORT values ('KEENE','23-SEP-01',99.8,82.6,NULL);
```

Questa riga invece inserisce Città, DataCampione (entrambe colonne NOT NULL) e Precipitazione in COMFORT:

```
insert into COMFORT (Citta, DataCampione, Precipitazione) values  
('KEENE','22-DEC-01',3.9);
```

Per copiare soltanto i dati della città di KEENE in una nuova tabella di nome NEW\_HAMPSHIRE si può utilizzare:

```
insert into NEW_HAMPSHIRE select * from COMFORT  
where Citta = 'KEENE';
```

Per informazioni riguardanti l'inserimento di righe all'interno di tabelle che utilizzano tipi di dati astratti, si *rimanda* ai Capitoli 4 e 30. Per dettagli sull'inserimento di righe in tabelle annidate e array variabili, si consulti il Capitolo 31.

## INSTEAD OF, TRIGGER

Un trigger INSTEAD OF esegue un blocco di codice PL/SQL al posto della transazione che provoca l'esecuzione del trigger. Solitamente, questi tipi di trigger vengono impiegati nel reindirizzamento delle transazioni dirette a viste. Vedere CREATE TRIGGER e i Capitoli 28 e 30.

## INSTR

**VEDERE ANCHE** CARATTERI, FUNZIONI; SUBSTR; Capitolo 7.

### SINTASSI

```
INSTR(stringa, set[, inizio [, occorrenza ]])
```

**DESCRIZIONE** INSTR trova la posizione di un *set* di caratteri in una *stringa*, partendo dalla posizione di *inizio* della stringa e cercando la prima, seconda, terza... *occorrenza* del *set*. Questa funzione opera anche con i tipi di dati NUMBER e DATE. Anche *inizio* può essere un numero negativo, in tal caso la ricerca comincia con i caratteri alla fine della stringa e procede all'indietro.

### ESEMPIO

Per trovare la terza occorrenza di 'PI' nella stringa seguente, si può utilizzare:

```
INSTR('PETER PIPER PICKED A PECK OF PICKLED PEPPERS','PI',1,3)
```

Il risultato di questa funzione è 30, la posizione della terza occorrenza di ‘PI’.

## INTEGRITÀ REFERENZIALE

L’integrità referenziale è la proprietà che garantisce che i valori di una colonna dipendono da quelli presenti in un’altra colonna. Questa proprietà viene imposta tramite vincoli di integrità. Vedere VINCOLO DI INTEGRITÀ.

## INTERSECT

**VEDERE ANCHE** MINUS, OPERATORI DI QUERY, UNION, Capitolo 12.

### SINTASSI

```
select...
INTERSECT
select...
```

**DESCRIZIONE** INTERSECT combina due query e restituisce solo le righe dalla prima istruzione select che sono identiche ad almeno una riga di quelle della seconda istruzione select. Tra le istruzioni select, il numero di colonne e i tipi di dati devono essere identici, condizione non richiesta per i nomi delle colonne. Tuttavia, per far sì che l’intersezione passi i dati, questi dovranno essere identici nelle righe prodotte.

## INTERVAL DAY TO SECOND

Il tipo di dati INTERVAL DAY (*precisione\_giorno*) TO SECOND (*precisione\_secondo*) memorizza un lasso di tempo espresso in giorni, ore, minuti e secondi. *precisione\_giorno* può essere specificata come il numero di cifre nel campo della data/ora GIORNO (il valore predefinito è 2), e *precisione\_secondo* come il numero di cifre nella parte frazionaria del campo Secondi (il valore predefinito è 6).

## INTERVAL YEAR TO MONTH

Il tipo di dati INTERVAL YEAR (*precisione*) TO MONTH memorizza un lasso di tempo espresso in anni e mesi, in cui *precisione* è il numero di cifre nel campo della data/ora ANNO (il valore predefinito è 2).

## INTESTAZIONE DI RIGA

È quella parte della riga che contiene le informazioni sulla riga che non riguardano i dati della riga, ossia il numero di campi, i tipi delle colonne e così via.

## IS NULL

**VEDERE ANCHE** OPERATORI LOGICI, Capitoli 3 e 8.

### SINTASSI

```
WHERE colonna IS [NOT] NULL
```

**DESCRIZIONE** IS NULL verifica che in una colonna (o in un'espressione) non vi siano dati. Un test di nullità è differente da un test di uguaglianza in quanto NULL significa che un valore è sconosciuto o irrilevante e perciò non si può dire che sia uguale a qualche altra cosa, ivi compreso un altro valore NULL.

## ISTANZA

Un'istanza è tutto ciò che serve per eseguire Oracle: processi in background (programmi), memoria e così via. Un'istanza è il mezzo attraverso cui si accede a un database.

## ISTANZA, IDENTIFICATIVO

Un identificativo d'istanza è lo strumento con cui è possibile distinguere un'istanza dall'altra quando su un host ne esistono diverse.

## ISTOGRAMMA

Un istogramma mostra la distribuzione dei dati di una colonna. L'ottimizzatore di Oracle può utilizzare gli istogrammi per determinare l'efficienza degli indici per particolari intervalli di valori. Vedere ANALYZE.

## JAVA

Java è un linguaggio di programmazione sviluppato in origine da Sun Microsystems. Gli oggetti procedurali memorizzati possono essere scritti sia con PL/SQL sia con Java. Vedere il Capitolo 34.

## JDBC

JDBC è l'acronimo di Java Database Connectivity, un'interfaccia di database per la programmazione di applicazioni industriali standard. Oracle supporta le connessioni JDBC. Vedere il Capitolo 35.

## JOIN

**VEDERE ANCHE** SELECT, Capitolo 3.

### SINTASSI

WHERE TableA.column = TableB.column

oppure:

```
FROM TableA INNER JOIN TableB  
  on TableA.Column = TableB.Column
```

oppure:

```
FROM TableA INNER JOIN TableB  
  using(Column)
```

oppure:

```
FROM TableA NATURAL JOIN TableB
```

**DESCRIZIONE** Un join combina colonne e dati da due o più tabelle (e raramente da parti diverse della stessa tabella). Le tabelle sono tutte elencate nella clausola `from` dell'istruzione `select` e la relazione tra le due tabelle può essere specificata nella clausola `where` generalmente attraverso una semplice egualanza del tipo:

```
where LIBRO_ORDINE.Titolo = BIBLIOTECA.Titolo
```

Questo costrutto viene spesso indicato col nome di *equi-join* poiché utilizza il segno di uguale nella clausola `where`. Si possono unire tramite join delle tabelle anche con altre forme di uguaglianza come `>=`, `<` e così via, che raramente danno risultati significativi. Molto spesso uno o entrambi i membri del segno di uguale possono contenere un'espressione, talvolta una `SUBSTR` oppure una combinazione di colonne, che viene utilizzata per stabilire l'uguaglianza con l'altro membro. Si tratta di un impiego abbastanza comune.

L'unione tramite join di due tabelle senza una clausola di join, specificata nella clausola `from` oppure nella clausola `where`, dà come risultato un prodotto cartesiano, che associa ogni riga in una tabella ad ogni riga di un'altra tabella. Una tabella di 80 righe combinata con un'altra di 100 righe produce un risultato di 8000 righe (in genere ben poco significativo).

Un *outer join* è un metodo che permette di recuperare intenzionalmente le righe di una tabella che non corrispondono a quelle di un'altra tabella. A partire da Oracle9i è possibile utilizzare le opzioni di sintassi degli outer join dello standard ANSI (LEFT OUTER JOIN, RIGHT OUTER JOIN e FULL OUTER JOIN). Per esempi sulla sintassi degli outer join, si rimanda al Capitolo 12.

## JOIN INTERNO

I join interni restituiscono delle righe in base ai valori chiave che due tabelle hanno in comune. Supportano sia la clausola `ON` sia la clausola `USING`. Per esempio, la query seguente unisce tramite join `LIBRO_ORDINE` a `BIBLIOTECA` basandosi sui valori della colonna `Titolo`:

```
select BO.Titolo
  from LIBRO_ORDINE BO inner join BIBLIOTECA B
    on BO.Titolo = B.Titolo;
```

Questa query può essere riscritta come:

```
select Titolo
  from LIBRO_ORDINE BO inner join BIBLIOTECA B
 using (Titolo);
```

## JOIN NATURALE

Un join naturale unisce due tabelle basandosi sulle colonne di chiave che esse hanno in comune. Di seguito ne è riportata la sintassi:

```
FROM TabellaA NATURAL JOIN TabellaB
```

che equivale a:

```
FROM TabellaA, TabellaB  
WHERE TabellaA.KeyColumn = TabellaB.KeyColumn
```

Per esempi dei tipi diversi di sintassi di join, si rimanda al Capitolo 12.

## KERNEL

Il kernel è il codice di programma fondamentale di Oracle RDBMS (una collezione di più moduli) chiamabile dai processi in background.

## LABEL (PL/SQL)

Un'etichetta è una parola associata a un'istruzione eseguibile, in genere per fungere da destinazione di un'istruzione GOTO.

## LAG

**VEDERE ANCHE** LEAD.

### SINTASSI

```
LAG ( valore_espr [, offset] [, predefinito] )  
OVER ( [ clausola_partizionamento_query ] clausola_order_by )
```

**DESCRIZIONE** LAG consente di accedere contemporaneamente a più righe di una tabella senza un self-join. Con una serie di righe restituite da una query e una posizione del cursore, LAG consente di accedere a una riga che si trova in un determinato offset fisico che precede questa posizione.

## LANCIARE

In Java, i programmi lanciano le eccezioni tramite la clausola try, e le elaborano con la clausole catch e finally. Vedere il Capitolo 34.

## LAST

**VEDERE ANCHE** DENSE\_RANK, FIRST.

### SINTASSI

```
funzione_aggregazione_ KEEP  
( DENSE_RANK LAST ORDER BY  
  expr [ DESC | ASC ] [ NULLS { FIRST | LAST } ]  
  [, expr [ DESC | ASC ] [ NULLS { FIRST | LAST } ]],...  
) [OVER clausola_partizionamento_query]
```

**DESCRIZIONE** LAST è sia una funzione di aggregazione sia una funzione analitica che opera su un gruppo di valori tratti da un gruppo di righe che compaiono per ultime rispetto a una determinata specifica di ordinamento. *funzione\_aggregazione* è una delle funzioni di gruppo numeriche standard (come MIN, MAX, AVG e STDDEV).

## **LAST\_DAY**

**VEDERE ANCHE** ADD\_MONTHS; DATE, FUNZIONI; Capitolo 9.

### **SINTASSI**

LAST\_DAY (*data*)

**DESCRIZIONE** LAST\_DAY fornisce la data dell'ultimo giorno del mese cui appartiene la *data* indicata.

### **ESEMPIO**

Questa istruzione:

```
LAST_DAY ('05.11.01')
```

produce come risultato 30.11.01.

## **LAST\_VALUE**

**VEDERE ANCHE** FIRST\_VALUE.

### **SINTASSI**

LAST\_VALUE ( *espr* ) OVER ( *clausola\_analitica* )

**DESCRIZIONE** LAST\_VALUE è una funzione analitica. Restituisce l'ultimo valore in un gruppo ordinato di valori.

## **LEAD**

**VEDERE ANCHE** LAG.

### **SINTASSI**

```
LEAD ( valore_espr [ , offset] [ , predefinito] )
      OVER ( [clausola_partizionamento_query] clausola_order_by )
```

**DESCRIZIONE** LEAD consente di accedere contemporaneamente a più righe di una tabella senza un self-join. Con una serie di righe restituite da una query e una posizione del cursore, LEAD consente di accedere a una riga che si trova in un offset fisico che segue questa posizione.

## **LEAST**

**VEDERE ANCHE** GREATEST, FUNZIONI DI ELENCO, Capitolo 9.

### **SINTASSI**

LEAST(*valore1*, *valore2*, ...)

**DESCRIZIONE** LEAST è il valore di un elenco di colonne, espressioni o valori. I valori possono essere di tipo VARCHAR2, CHAR, DATE o NUMBER, sebbene LEAST non valuti correttamente le date in formato letterale (come '20-MAY-49') se non con l'ausilio della funzione

TO\_DATE. Per una discussione relativa alla valutazione dei numeri negativi, si consulti GREATEST.

## LENGTH

**VEDERE ANCHE** CARATTERI, FUNZIONI; VSIZE; Capitolo 7.

### SINTASSI

LENGTH(*stringa*)

**DESCRIZIONE** LENGTH indica la lunghezza di una stringa, di un numero, di una data o di una espressione.

## LEVEL

**VEDERE ANCHE** CONNECT BY, PSEUDOCOLONNE, Capitolo 13.

### SINTASSI

LEVEL

**DESCRIZIONE** Level è una pseudocolonna, utilizzata con CONNECT BY, il cui valore è 1 nel nodo principale, 2 per un figlio di primo livello, 3 per un figlio di secondo livello e così via. Level fornisce un'indicazione di quanti livelli dell'albero sono stati analizzati.

## LGWR

LGWR (LoG WRiter process) scrive voci redo log dalla SGA nei redo log in linea. *Vedere* BACKGROUND, PROCESSO.

## LIBRERIA

I blocchi PL/SQL possono fare riferimento a sottoprogrammi esterni, memorizzati in librerie. *Vedere* CREATE LIBRARY.

## LIKE

**VEDERE ANCHE** OPERATORI LOGICI, Capitolo 3.

### SINTASSI

WHERE *stringa* LIKE *stringa*

**DESCRIZIONE** LIKE esegue una ricerca per modelli. Un carattere di sottolineatura rappresenta uno e un solo spazio. Un segno di percentuale rappresenta un numero qualsiasi di spazi o caratteri, incluso il valore zero. Se LIKE ha un \_ o un % nella prima posizione di un confronto (come nel secondo e nel terzo degli esempi proposti) qualsiasi indice sulla colonna viene ignorato.

**ESEMPI**

Argomento LIKE 'M0%' "Argomento inizia con le lettere M0."  
 Argomento LIKE '\_I%' "Argomento ha una I in terza posizione."  
 Argomento LIKE '%0%0%' "Argomento contiene almeno due 0."

**LIMITAZIONE**

Una regola o una limitazione riguardante una parte di dati (come una limitazione NOT NULL su una colonna) che viene imposta a livello di dati, anziché a livello di applicazione o di oggetto.  
*Vedere VINCOLO DI INTEGRITÀ.*

**LINESIZE (SQL\*PLUS)**

*Vedere SET.*

**LIST**

**VEDERE ANCHE** APPEND, CHANGE, EDIT, INPUT, RUN, Capitolo 6.

**SINTASSI**

`L[IST] [ {inizio}*] [fine]* ]`

**DESCRIZIONE** LIST elenca le linee del buffer corrente a partire da *inizio* fino a *fine*. Gli estremi sono entrambi numeri interi. La riga finale diventa la riga corrente del buffer e viene evidenziata con un asterisco. LIST senza parametri elenca tutte le linee. Un asterisco in una delle due posizioni la imposta alla linea corrente. LIST con un solo parametro visualizza solo la linea richiesta, mentre LIST \* visualizza la sola linea corrente. Lo spazio tra LIST e *inizio* non è necessario ma aiuta la leggibilità.

**ESEMPIO**

Per elencare il buffer corrente di SQL, si utilizzi quanto segue:

LIST

Un asterisco indica la linea corrente. Per elencare solo la seconda linea, si digiti:

LIST 2

Con questa chiamata anche 2 si trasforma nella linea del buffer corrente.

**LN**

**VEDERE ANCHE** NUMERI, FUNZIONI ; Capitolo 8.

**SINTASSI**

`LN(numero)`

**DESCRIZIONE** LN è il logaritmo “naturale” o in base *e*, di un numero.

## LOADJAVA

LOADJAVA è un utility che serve per caricare le classi, le risorse e i sorgenti Java nel database. Se si intende utilizzare le classi Java nelle stored procedure, le si dovrà caricare nel database. Per rimuovere le classi Java dal database, si utilizzi l'utility DROPJAVA.

## LOB

Un LOB è un oggetto molto grande. Oracle supporta diversi tipi di dati di grandi dimensioni, tra cui i BLOB (binary large object), i CLOB (character large object) e i BFILE (binary file, memorizzati all'esterno del database). Vedere il Capitolo 32 e la clausola LOB di CREATE TABLE.

## LOBOFFSET (SQL\*PLUS)

Vedere SET.

## LOBSOURCE (SQL\*PLUS)

Vedere SET.

## LOCALTIMESTAMP

**VEDERE ANCHE** DATE, FUNZIONI; Capitolo 9.

### SINTASSI

LOCALTIMESTAMP [( *precisione\_indicatore\_orario* )]

**DESCRIZIONE** LOCALTIMESTAMP restituisce la data o l'orario correnti nel fuso orario della sessione come un valore del tipo di dati TIMESTAMP.

### ESEMPIO

SELECT LOCALTIMESTAMP FROM DUAL;

## LOCK TABLE

**VEDERE ANCHE** COMMIT, DELETE, INSERT, ROLLBACK, SAVEPOINT, UPDATE.

### SINTASSI

```
LOCK TABLE
[schema .] { tabella | vista }
[ { PARTITION ( partizione ) | SUBPARTITION ( partizione_secondaria ) }
| @ dblink ]
[, [schema .] { tabella | vista }
[ { PARTITION ( partizione ) | SUBPARTITION ( partizione_secondaria ) }
| @ dblink ] ]...
IN modalità_lock MODE [NOWAIT];
```

**DESCRIZIONE** LOCK TABLE blocca una tabella con una delle modalità specificate che ne permettono la condivisione senza compromettere l'integrità dei dati. L'uso di LOCK TABLE permette di concedere ad altri utenti l'accesso continuato ma parziale alla tabella. A prescindere dall'opzione scelta, la tabella resta nella modalità di lock richiesta fino a che non si effettua un commit o un rollback sulle proprie transazioni.

Le modalità di lock comprendono ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE, SHARE, SHARE ROW EXCLUSIVE ed EXCLUSIVE.

I blocchi di tipo EXCLUSIVE consentono agli utenti di eseguire solamente la query sulla tabella bloccata e null'altro. Nessun altro utente può bloccare la tabella. I blocchi SHARE permettono query concorrenti ma vietano qualsiasi aggiornamento della tabella bloccata.

Con un blocco di tipo ROW SHARE o SHARE UPDATE, nessun utente può bloccare l'intera tabella per accedervi esclusivamente. Di fatto si permettono accessi concorrenti alla tabella da parte di tutti gli utenti. I due tipi di blocchi sono sinonimi: SHARE UPDATE esiste solo per compatibilità con versioni precedenti di Oracle.

I blocchi ROW EXCLUSIVE sono simili a quelli di tipo ROW SHARE ma vietano i blocchi condivisi (per cui un solo utente per volta può accedere alla tabella).

Se un comando LOCK TABLE ha dei problemi a portare a termine il proprio compito (in genere quando qualcun altro ha eseguito un comando analogo quasi nello stesso momento) aspetterà fino alla conclusione della situazione concorrente. Se si vogliono evitare problematiche di questo tipo e si desidera che il controllo torni al programma occorre utilizzare l'opzione NOWAIT. Si noti che è possibile bloccare partizioni e partizioni secondarie specifiche.

## LOG

**VEDERE ANCHE** NUMERI, FUNZIONI; Capitolo 8.

### SINTASSI

LOG(*base*, *numero*)

**DESCRIZIONE** LOG fornisce il logaritmo di un *numero* per una *base* specifica.

### ESEMPIO

```
LOG(EXP(1),3) = 1.098612 --log(e) di 3
LOG(10,100)   = 2          --log(10) di 100
```

## LOG DELLE VISTE MATERIALIZZATE

Quando si aggiorna una vista materializzata, si può eseguire un aggiornamento completo oppure uno incrementale. Un aggiornamento incrementale, chiamato anche aggiornamento “rapido” invia solo le modifiche DML dalla tabella master alla vista materializzata. Per tenere traccia delle modifiche apportate alla tabella master della vista materializzata, occorre creare un log della vista materializzata. Il log della vista materializzata deve trovarsi nello stesso schema di quello della tabella master della vista materializzata. Gli aggiornamenti rapidi non sono disponibili per le viste materializzate complesse. Vedere CREATE MATERIALIZED VIEW LOG e il Capitolo 23.

## LOG WRITER PROCESS (LGWR)

Vedere LGWR.

## LOGIN ACCOUNT

Un login account è un'accoppiata di nome utente e password che permette agli utenti di utilizzare l'RDBMS di Oracle. In genere, questo account è distinto da quello del sistema operativo.

## LONG (SQL\*PLUS)

Vedere SET.

## LONG RAW, TIPO DI DATI

L'unica cosa che distingue una colonna LONG da una di tipo LONG RAW è che la seconda contiene dati binari di tipo RAW. Nelle colonne LONG RAW i valori devono essere inseriti in notazione esadecimale.

## LOWER

**VEDERE ANCHE** CARATTERI, FUNZIONI; INITCAP; UPPER; Capitolo 7.

### SINTASSI

LOWER(*stringa*)

**DESCRIZIONE** LOWER converte ciascuna lettera di una stringa in caratteri minuscoli.

### ESEMPIO

LOWER('Penisola') = penisola

## LPAD

**VEDERE ANCHE** CARATTERI, FUNZIONI; LTRIM; RPAD; RTRIM; Capitolo 7.

### SINTASSI

LPAD(*stringa*, *lunghezza* [, 'set'])

**DESCRIZIONE** LPAD fa in modo che una *stringa* raggiunga una determinata *lunghezza* aggiungendo un certo *set* di caratteri a sinistra della stringa. Se non si specifica *set*, il carattere di riempimento predefinito è lo spazio.

### ESEMPIO

\_LPAD('>', 11, '-')

produce

- - - - >

## LTRIM

**VEDERE ANCHE** CARATTERI, FUNZIONI; LPAD; RPAD; RTRIM; Capitolo 7.

### SINTASSI

`LTRIM(stringa [, 'set'])`

**DESCRIZIONE** LTRIM taglia tutte le occorrenze (anche parziali) di un *set* di caratteri dall'estremità sinistra di una *stringa*.

### ESEMPIO

`LTRIM('NANCY', 'AN')`

produce:

CY

## MAKE\_REF

La funzione MAKE\_REF costruisce un REF (riferimento) dalla chiave esterna di una tabella che si riferisce alla tabella base di una vista oggetto. MAKE\_REF permette di sovrapporre riferimenti a relazioni di chiave esterna preesistenti. *Vedere* il Capitolo 33.

## MAX

**VEDERE ANCHE** COMPUTE, FUNZIONI DI GRUPPO, MIN Capitolo 8.

### SINTASSI

`MAX([DISTINCT | ALL] valore)`

**DESCRIZIONE** MAX è il massimo di tutti i *valori* di un gruppo di righe. MAX ignora la presenza di valori NULL. L'opzione DISTINCT non è significativa dato che il massimo di tutti i valori coincide con il massimo dei valori separati.

## METODO

Un metodo è un blocco di codice. Con riferimento ai tipi di dati astratti, un metodo è un blocco di codice PL/SQL utilizzato per incapsulare i metodi di accesso ai dati di un oggetto. I metodi sono specificati come parte della specifica dei tipi di dati astratti (*vedere* CREATE TYPE) mentre il loro corpo è specificato come parte del comando CREATE TYPE BODY. Gli utenti possono eseguire i metodi sui tipi di dati per cui sono stati definiti. Il constructor method creato per ciascun tipo di dati astratto è un esempio di metodo. Per esempi di metodi si *consulti* il Capitolo 30.

In Java, un metodo è un membro di una classe. Si può chiamare il metodo di una classe e passargli dei parametri in modo che li possa usare quando esegue i propri comandi. Per esempi, si consulti il Capitolo 30.

## MIN

**VEDERE ANCHE** COMPUTE, FUNZIONI DI GRUPPO, MAX, Capitolo 8.

## SINTASSI

`MIN([DISTINCT | ALL] valore)`

**DESCRIZIONE** MIN è il minimo di tutti i *valori* di un gruppo di righe. MIN ignora la presenza di valori NULL. L'opzione DISTINCT non è significativa dato che il minimo di tutti i valori è identico al minimo di valori presi distintamente.

## MINUS

**VEDERE ANCHE** INTERSECT, OPERATORI DI QUERY, UNION, OPERATORI DI RICERCA TESTUALE, Capitoli 12 e 24.

### SINTASSI

```
select
MINUS
select
```

in Oracle Text le query eseguite sugli indici CONTEXT:

```
select colonna
  from TABELLA
 where CONTAINS(Text,'text MINUS text') >0;
```

**DESCRIZIONE** MINUS combina due query. Restituisce solo le righe dalla prima istruzione select che non sono prodotte dalla seconda istruzione select (il risultato della prima MENO il risultato della seconda). Il numero di colonne e i tipi di dati devono essere identici tra le istruzioni select, condizione non richiesta per i nomi delle colonne. Tuttavia, i dati devono essere uguali nelle righe prodotte perché MINUS li possa rifiutare. Per una discussione sulle differenze importanti e sugli effetti di INTERSECT, MINUS e UNION, si *consulti* il Capitolo 12

Nelle ricerche testuali eseguite sugli indici CONTEXT, MINUS indica alla ricerca di testo di due termini di sottrarre il punteggio ottenuto dalla ricerca del secondo termine da quello della ricerca del primo, prima di confrontare il risultato con il punteggio di soglia.

## MOD

**VEDERE ANCHE** NUMERI, FUNZIONI; Capitolo 8.

### SINTASSI

`MOD(valore, divisore)`

**DESCRIZIONE** MOD divide un *valore* per un *divisore* e fornisce il resto.  $MOD(23,6) = 5$  significa dividere 23 per 6. Il 6 nel 23 sta 3 volte col resto di 5, e questo è il risultato del modulo. *valore* e *divisore* possono essere numeri reali qualsiasi, ma il *divisore* non può essere uguale a 0.

### ESEMPI

<code>MOD(100,10)</code>	=	0
<code>MOD(22,23)</code>	=	22
<code>MOD(10,.3)</code>	=	1
<code>MOD(-30.23,7)</code>	=	-2.23
<code>MOD(4.1,.3)</code>	=	.2

Il secondo esempio mostra il comportamento di MOD qualora il divisore sia più grande del dividendo (il numero da dividere). Il risultato è il dividendo stesso. Si noti anche il caso importante:

`MOD(valore,1) = 0`

è vera se *valore* è un numero intero. Questo è un ottimo trucco per verificare che un numero sia effettivamente un intero.

## **MONTHS\_BETWEEN**

**VEDERE ANCHE** ADD\_MONTHS; DATE, FUNZIONI; Capitolo 9.

### **SINTASSI**

`MONTHS_BETWEEN(data2,data1)`

**DESCRIZIONE** `MONTHS_BETWEEN` fornisce la differenza tra la *data2* e la *data1* in mesi. In genere il risultato non è un numero intero.

## **MOUNT DI UN DATABASE**

L'esecuzione del MOUNT di un database equivale a renderlo disponibile all'amministratore del database.

## **MOUNT E OPEN DI UN DATABASE**

Il MOUNT e l'OPEN di un database consentono di renderlo disponibile per gli utenti.

## **NCHAR**

NCHAR è una versione a più byte del tipo di dati CHAR. Memorizza dati carattere a lunghezza fissa fino a 2.000 byte di lunghezza. *Vedere DATI, TIPI.*

## **NCHR**

**VEDERE ANCHE** CHR.

### **SINTASSI**

`NCHR(numero)`

**DESCRIZIONE** NCHR restituisce il carattere che nel set di caratteri nazionale possiede l'equivalente binario di *numero*.

## **NCLOB**

Il tipo di dati NCLOB, è la versione a più byte del tipo di dati CLOB. NCLOB memorizza oggetti di grandi dimensioni che contengono caratteri UNICODE, fino a una lunghezza massima di 4GB. *Vedere DATI, TIPI.*

## NEAR

**VEDERE ANCHE** CONTAINS, Capitolo 24.

**DESCRIZIONE** Negli indici CONTEXT, NEAR indica di eseguire una ricerca di prossimità per le stringhe di testo specificate. Se i termini da cercare fossero ‘estate’ e ‘vacanza’, una ricerca di prossimità potrebbe essere strutturata nella maniera seguente:

```
select Testo  
  from SONNET  
 where CONTAINS(Testo, 'estate NEAR vacanza') >0;
```

Quando si valutano i risultati della ricerca testuale, le righe contenenti ‘estate’ e ‘vacanza’ in stretta prossimità ottengono punteggi maggiori di quelle in cui le due parole distano di più.

## NEWPAGE (SQL\*PLUS)

*Vedere* SET.

## NEW\_TIME

*Vedere* DATE, FUNZIONI.

## NEXT\_DAY

*Vedere* DATE, FUNZIONI.

## NEXTVAL

*Vedere* PSEUDOCOLONNE.

## NLS\_CHARSET\_DECL\_LEN

**VEDERE ANCHE** NCHAR.

### SINTASSI

```
NLS_CHARSET_DECL_LEN (conta_byte, id_set_car)
```

**DESCRIZIONE** NLS\_CHARSET\_DECL\_LEN restituisce la larghezza della dichiarazione (espressa in numeri di caratteri) di una colonna di tipo NCHAR.

## NLS\_CHARSET\_ID

**VEDERE ANCHE** NLS\_CHARSET\_NAME.

### SINTASSI

```
NLS_CHARSET_ID ( testo )
```

**DESCRIZIONE** NLS\_CHARSET\_ID restituisce il numero di ID del set di caratteri che corrisponde a *testo* del nome del set di caratteri.

## NLS\_CHARSET\_NAME

**VEDERE ANCHE** NLS\_CHARSET\_ID.

### SINTASSI

`NLS_CHARSET_NAME ( numero )`

**DESCRIZIONE** NLS\_CHARSET\_NAME restituisce il nome del set di caratteri che corrisponde a *numero* del numero di ID.

## NLS\_INITCAP

**VEDERE ANCHE** CARATTERI, FUNZIONI, INITCAP, Capitolo 8.

### SINTASSI

`NLS_INITCAP(numero[, parametri_nls])`

**DESCRIZIONE** NLS\_INITCAP è analoga alla funzione INITCAP tranne per l'aggiunta di una stringa di parametri. La stringa *parametri NLS* racchiusa tra apici fornisce una sequenza di ordinamento per rendere maiuscole alcune sequenze di caratteri particolari di alcuni linguaggi internazionali. INITCAP verrà utilizzata con la sequenza di ordinamento predefinita per la sessione, tuttavia questa funzione consente di specificare l'esatta sequenza di ordinamento da utilizzare.

## NLS\_LOWER

**VEDERE ANCHE** CARATTERI, FUNZIONI; LOWER; Capitolo 8.

### SINTASSI

`NLS_LOWER(numero[, parametri_nls])`

**DESCRIZIONE** NLS\_LOWER è analoga alla funzione LOWER salvo per l'aggiunta di una stringa di parametri. La stringa *parametri NLS* racchiusa tra apici fornisce una sequenza di ordinamento per rendere minuscole delle sequenze di caratteri particolari di alcuni linguaggi internazionali. LOWER verrà utilizzata con la sequenza di ordinamento predefinita per la sessione, tuttavia questa funzione consente di specificare l'esatta sequenza di ordinamento da utilizzare.

## NLS\_UPPER

**VEDERE ANCHE** CARATTERI, FUNZIONI; UPPER; Capitolo 8.

### SINTASSI

`NLS_UPPER(numero[, parametri_nls])`

**DESCRIZIONE** NLS\_UPPER è analoga alla funzione UPPER salvo per l'aggiunta di una stringa di parametri. La stringa *parametri NLS* racchiusa tra apici fornisce una sequenza di ordinamento.

mento per rendere maiuscole alcune sequenze di caratteri particolari di alcuni linguaggi internazionali. UPPER verrà utilizzata con la sequenza di ordinamento predefinita per la sessione, tuttavia questa sessione consente di specificare l'esatta sequenza di ordinamento da utilizzare.

## NLSSORT

**VEDERE ANCHE** Oracle9i Database Administrator's Guide, Capitolo 37.

### SINTASSI

NLSSORT(*carattere[.parametri\_nls]*)

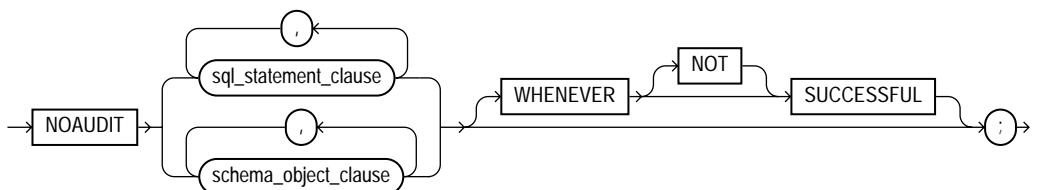
**DESCRIZIONE** NLSSORT fornisce il valore (intero) della sequenza di ordinamento del *carattere* specificato basato sull'opzione National Language Support selezionata per il sito. La stringa *parametri NLS* racchiusa tra apici fornisce una sequenza di ordinamento per sequenze linguistiche particolari, se non si desidera ricorrere all'impostazione predefinita per la sessione o il database.

## NOAUDIT

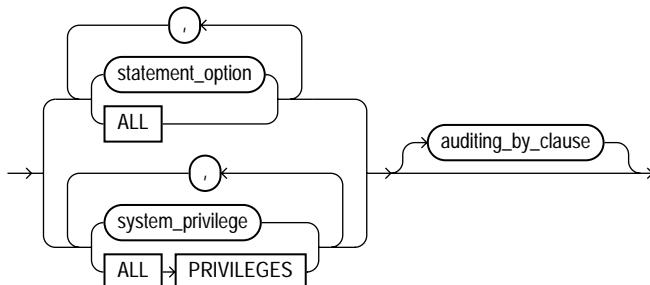
**VEDERE ANCHE** AUDIT, PRIVILEGI.

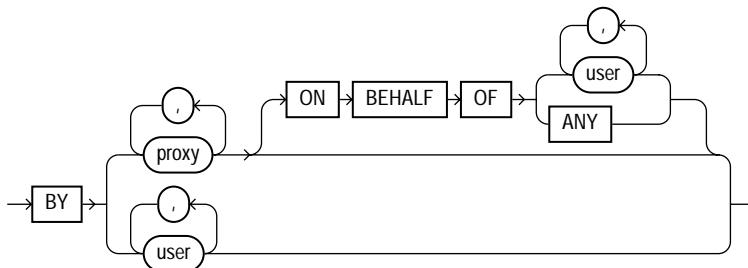
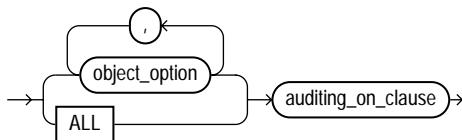
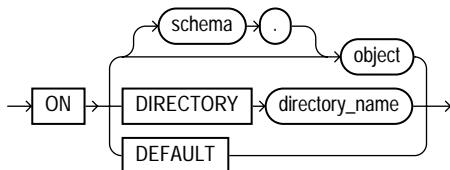
### SINTASSI

noaudit



### sql\_statement\_clause



**auditing\_by\_clause****schema\_object\_clause****auditing\_on\_clause**

**DESCRIZIONE** NOAUDIT interrompe l’audit delle istruzioni SQL tenute sotto controllo a seguito del comando AUDIT. Interrompe l’audit sia di istruzioni semplici (*vedere AUDIT*) sia di quelle autorizzate da privilegi di sistema (*vedere PRIVILEGI*). Se è presente la clausola BY e una lista di utenti, il comando interrompe l’audit delle istruzioni da essi impartite. In caso contrario, Oracle interrompe l’audit delle istruzioni impartite da tutti gli utenti. L’opzione WHENEVER SUCCESSFUL interrompe l’audit solo per le istruzioni completate con successo; WHENEVER NOT SUCCESSFUL disabilita l’audit delle istruzioni che generano errori.

NOAUDIT interrompe anche l’audit di un’opzione per l’uso di una tabella, di una vista o di un sinonimo. A tal fine, si devono possedere gli oggetti appena elencati o avere il privilegio DBA. *opzione* è una delle opzioni descritte di seguito. *utente* è il nome utente del proprietario dell’oggetto. *oggetto* è una tabella, una vista o un sinonimo. *opzione* specifica per quali comandi interrompere l’audit. Per una tabella o una vista materializzata le opzioni sono ALTER, AUDIT, COMMENT, DELETE, GRANT, INDEX, INSERT, LOCK, RENAME, SELECT e UPDATE. Per le procedure, le funzioni, i tipi, le librerie e i package si possono tenere sotto controllo i comandi EXECUTE e GRANT. Per le directory si possono tenere sotto controllo AUDIT, GRANT e READ. GRANT tiene sotto controllo il comando GRANT e il comando REVOKE. NOAUDIT GRANT li interrompe entrambi. ALL interrompe l’audit di tutti questi.

ON *oggetto* assegna un nome agli oggetti controllati e include tabelle, viste oppure sinonimi di tabelle, viste o sequenze.

Non è possibile utilizzare ALTER e INDEX per le viste. Le opzioni predefinite delle viste sono create dall’unione delle opzioni di ciascuna delle tabelle base più le opzioni DEFAULT.

Per i sinonimi, le opzioni sono le stesse delle tabelle.

Per le sequenze le opzioni sono ALTER, AUDIT, GRANT e SELECT.

**WHENEVER SUCCESSFUL** disattiva l'audit degli aggiornamenti delle tabelle che hanno avuto successo. **WHENEVER NOT SUCCESSFUL** disattiva l'audit degli aggiornamenti non conclusi. Se si omette questa clausola facoltativa, entrambi i tipi di audit vengono disattivati.

Entrambe le forme concludono qualsiasi operazione aperta sul database. Se l'accesso alle tabelle remote avviene tramite un database link, qualsiasi audit sul sistema remoto è basato su opzioni impostate all'interno di quest'ultimo. Le informazioni di audit sono riportate nella tabella SYS.AUD\$. Per dettagli sulle viste di audit, si consulti il Capitolo 37.

### ESEMPI

La riga seguente interrompe l'audit di tutti i tentativi di aggiornamento o cancellazione della tabella BIBLIOTECA:

```
noaudit update, delete on BIBLIOTECA;
```

Questa riga interrompe l'audit di tutti gli accessi alla tabella BIBLIOTECA che non abbiano avuto successo:

```
noaudit all on BIBLIOTECA whenever not successful;
```

### NODO

Esistono due definizioni di nodo:

- Nelle tabelle strutturate ad albero un nodo non è altro che una riga.
- In una rete, un nodo è il punto fisico nella rete a cui viene collegato un computer.

### NODO TERMINALE

In una tabella strutturata ad albero, un nodo terminale è una riga che non ha alcuna riga figlia. È equivalente a una foglia.

### NOLOGGING

La clausola NOLOGGING nella creazione di oggetti (tabelle, LOB, indici, partizioni e così via) specifica che questa operazione non verrà trascritta nei redo log file in linea; per questo motivo non sarà recuperabile dopo un errore di disco. La creazione di oggetti non viene registrata, così come alcuni tipi di transazioni sugli oggetti. NOLOGGING occupa lo spazio della parola chiave UNRECOVERABLE introdotta con Oracle7.2. Si faccia riferimento ai Capitoli 21 e 40.

### NOME DI SERVIZIO

Un nome di servizio, che si trova nel file tnsnames.ora, è un alias per un database cui si può accedere con Oracle Net. Nel file tnsnames.ora, il nome di servizio viene associato a un host, a una porta o a un'istanza. Quindi, è possibile modificare il nome di servizio di un database senza cambiare il nome del database e si può anche spostare un database in un host diverso senza modificare il suo nome di servizio.

## NOMI DEGLI OGGETTI

Gli oggetti di un database devono necessariamente possedere un nome. Questo vale per tabelle, viste, sinonimi, alias, colonne, indici, utenti, tablespace e così via. Esistono alcune regole che vincolano la denominazione degli oggetti.

- Il nome di un oggetto può avere una lunghezza compresa tra 1 e 30 caratteri, tranne per i nomi di database, che sono lunghi fino a otto caratteri, e i nomi dei file di host, la cui lunghezza dipende dal sistema operativo.
  - I nomi non possono contenere virgolette.
  - Un nome deve iniziare con una lettera, contenere solo i caratteri A-Z, 0-9, \$, #, e \_, non deve essere una parola riservata di Oracle (vedere PAROLE RISERVATE), non deve duplicare il nome di un altro oggetto del database posseduto dallo stesso utente.
- Lettere maiuscole o minuscole non fanno differenza nei nomi degli oggetti. I nomi degli oggetti dovrebbero seguire una convenzione di denominazione sensata, come discusso nel Capitolo 2.

## NON-EQUI-JOIN

Un non-equi-join è una condizione di join diversa da quella di uguaglianza (=). Vedere EQUI-JOIN.

## NOT

**VEDERE ANCHE** OPERATORI LOGICI, Capitolo 3.

**DESCRIZIONE** NOT precede e completa l'effetto di ciascuno dei seguenti operatori logici: BETWEEN, IN, LIKE ed EXISTS. NOT può precedere anche NULL come in IS NOT NULL.

## NOT EXISTS

**VEDERE ANCHE** ANY, ALL, EXISTS, IN, Capitolo 12.

### SINTASSI

```
select ...
where NOT EXISTS (select...);
```

**DESCRIZIONE** NOT EXISTS restituisce un valore falso in una clausola where se la subquery che lo segue restituisce una o più righe. La clausola select della subquery può essere una colonna, un letterale o un asterisco, non ha importanza. L'unica parte importante è se la clausola where della subquery restituisce oppure no una riga.

### ESEMPIO

NOT EXISTS viene utilizzato spesso per determinare quali record di una tabella non hanno record corrispondenti in un'altra. Per esempi dell'uso di NOT EXISTS si rimanda al Capitolo 12.

## NTILE

**VEDERE ANCHE** WIDTH\_BUCKET.

## SINTASSI

```
NTILE ( espr ) OVER ( [clausola_partizionamento_query] clausola_order_by )
```

**DESCRIZIONE** NTILE è una funzione analitica. Divide un gruppo di dati ordinati in un numero di bucket indicato da un'espressione e assegna il numero di bucket adeguato a ogni riga. I bucket sono numerati e l'espressione deve dare come risultato una costante positiva per ciascuna partizione.

## NULL (Forma 1, PL/SQL)

**VEDERE ANCHE** BLOCCO, STRUTTURA.

### SINTASSI

```
NULL;
```

**DESCRIZIONE** L'istruzione NULL non ha nulla a che vedere con i valori NULL. Il suo scopo principale è rendere più leggibile una sezione di codice dicendo, letteralmente, di "non fare nulla". È anche un espediente per creare blocchi nulli (dato che PL/SQL vuole almeno un'istruzione eseguibile tra BEGIN ed END). In genere viene utilizzato come l'istruzione successiva (solitamente l'ultima) a una di una serie di test condizionali.

### ESEMPIO

```
IF Eta > 65 THEN
...
ELSEIF ETA BETWEEN 21 and 65 THEN
...
ELSE
NULL;
ENDIF;
```

## NULL (Forma 2, Valore di colonna SQL)

**VEDERE ANCHE** CREATE TABLE, FUNZIONI DI GRUPPO, VARIABILE INDICATORE, NVL, Capitolo 3.

**DESCRIZIONE** Un valore NULL è un valore sconosciuto, irrilevante o non significativo. Quaunque tipo di dati Oracle può essere NULL. In altre parole, qualsiasi colonna Oracle su una certa riga può essere priva di valori (a meno che la tabella non sia stata creata specificando l'opzione NOT NULL su quella colonna). NULL nei tipi di dati NUMBER non equivale a zero.

Pochi linguaggi procedurali consentono l'uso diretto di variabili contenenti valori NULL o sconosciuti, anche se Oracle e SQL operano questo supporto in maniera molto intuitiva. Per estendere i linguaggi per i quali Oracle ha sviluppato i precompilatori allo scopo di supportare il concetto NULL, si ricorre a una variabile indicatore associata a una variabile host. In pratica si tratta di una sorta di flag che indica se la variabile host è nulla o meno. La variabile host e la sua variabile indicatore possono essere indicate e impostate separatamente nel codice del linguaggio host ma risultano sempre concatenate in SQL o in PL/SQL.

La funzione NVL può servire a rilevare l'assenza di valori su una colonna e a convertire i valori NULL in un valore significativo del tipo di dati della colonna. Per esempio, NVL(Nome,'SCONOSCIUTO') converte nella stringa "SCONOSCIUTO" un valore NULL nella colonna Nome.

Nel caso di valori non NULL (ovvero, quando è presente un nome) la funzione NVL restituisce semplicemente il nome. NVL funziona in maniera analoga con numeri e date.

Con l'eccezione di COUNT(\*) e COMPUTE NUMBER, le funzioni di gruppo ignorano i valori nulli. Altre funzioni, quando riscontrano un valore NULL tra i dati in corso di valutazione, restituiscono un valore nullo. Si consideri l'esempio seguente:

```
NULL + 1066 is NULL.  
LEAST(NULL,'A','Z') is NULL.
```

Dato che NULL rappresenta un valore sconosciuto, due colonne NULL non sono uguali. Perciò il segno di uguale e gli altri operatori logici (tranne IS NULL e IS NOT NULL) non funzionano con valori NULL. Per esempio, questo listato:

```
where Nome = NULL
```

non è una clausola where valida. NULL richiede necessariamente la parola chiave IS:

```
where Nome IS NULL
```

Durante gli ordinamenti di tipo order by, i valori NULL precedono sempre tutti gli altri negli ordinamenti crescenti e vengono per ultimi in quelli decrescenti.

Quando si memorizzano valori NULL in un database, essi vengono rappresentati con un unico byte se ricadono tra due colonne di valori reali e con nessun byte se cadono al fondo di una riga (oltre il margine di colonna definito da create table). Se alcune colonne di una tabella sono destinate a contenere spesso valori NULL conviene raggrupparle verso il fondo dell'elenco di colonne di create table; in questo modo si riesce a risparmiare dello spazio su disco.

I valori NULL non compaiono negli indici; l'unico caso contrario è quello in cui tutti i valori di una chiave di clusterizzazione sono NULL.

## **NULLIF**

**VEDERE ANCHE** COALESCE, DECODE, NULL, Capitolo 17.

### **SINTASSI**

```
NULLIF ( expr1 . expr2 )
```

**DESCRIZIONE** NULLIF confronta *expr1* e *expr2*. Se sono uguali, la funzione restituisce NULL. Altrimenti, la funzione restituisce *expr1*. Non si può specificare il letterale NULL per *expr1*.

La funzione NULLIF è logicamente equivalente alla seguente espressione CASE:

```
CASE WHEN expr1 = expr2 THEN NULL  
ELSE expr1 END
```

## **NUMBER, TIPO DI DATI**

Un tipo di dati NUMBER è un tipo di dati standard di Oracle in grado di contenere un numero, con o senza punto decimale e segno. I valori validi sono lo zero e i numeri positivi e negativi nell'intervallo da 1.0E-130 a 9.99.. E125.

## NUMERICI, FORMATI

**VEDERE ANCHE** COLONNA, Capitolo 14.

**DESCRIZIONE** Queste opzioni funzionano sia con il comando `set numformat` che con `column format`:

9999990	Il conteggio dei nove e zero determina le cifre massime che possono essere visualizzate.
999,999,999.99	Le virgole e i decimali vengono specificati secondo il modello mostrato.
999990	Visualizza zero se il valore è zero.
099999	Visualizza i numeri con degli zeri iniziali.
\$99999	Un segno di dollaro inserito all'inizio di ciascun numero.
B99999	Non viene visualizzato niente se il valore è zero. Questo è il valore predefinito.
99999MI	Se il numero è negativo, un segno meno segue il numero. Per default il segno negativo è sulla sinistra.
S9999	Restituisce "+" per i valori positivi, "-" per quelli negativi.
99999PR	I numeri negativi vengono visualizzati compresi tra < e >.
99D99	Visualizza il decimale nella posizione indicata.
9G999	Visualizza il separatore di gruppo nella posizione indicata.
C9999	Visualizza il simbolo valutario ISO nella posizione indicata.
L999	Visualizza il simbolo di valuta locale.
,	Visualizza una virgola.
.	Visualizza un punto.
9.999EEEE	La rappresentazione avviene in notazione scientifica (devono esserci esattamente 4 E).
999V99	Moltiplica il numero per $10^n$ , dove $n$ rappresenta il numero di cifre alla destra di V. 999V99 trasforma 1234 in 123400.
RN	Visualizza il numerale romano per i numeri compresi tra 1 e 3999.
DATE	Visualizza il valore come data nel formato MM/DD/YY per le colonne NUMBER utilizzate per memorizzare date del calendario Giuliano.

## NUMERI, FUNZIONI

Di seguito è riportato un elenco ordinato di tutte le funzioni numeriche a singolo valore supportate attualmente dal linguaggio SQL di Oracle. Ciascuna viene riportata altrove in questa guida, assieme alla sintassi e ad esempi d'uso. Ogni funzione può essere utilizzata come una funzione SQL normale così come una funzione PL/SQL. Si consultino anche FUNZIONI DI GRUPPO e FUNZIONI DI ELENCO.

FUNZIONE	DEFINIZIONE
<i>valore1</i> + <i>valore2</i>	Addizione.
<i>valore1</i> - <i>valore2</i>	Sottrazione.
<i>valore1</i> * <i>valore2</i>	Moltiplicazione.
<i>valore1</i> / <i>valore2</i>	Divisione.
<b>ABS(<i>valore</i>)</b>	Valore assoluto.
<b>ACOS(<i>valore</i>)</b>	Arcocoseno di <i>valore</i> , in radianti.
<b>ASIN(<i>valore</i>)</b>	Arcoseno di <i>valore</i> , in radianti.
<b>ATAN(<i>valore</i>)</b>	Arcotangente di <i>valore</i> , in radianti.
<b>BITAND(<i>valore1</i>, <i>valore2</i>)</b>	AND BIT per BIT di <i>valore1</i> e <i>valore2</i> : entrambi devono dare come risultato interi non negativi e restituire un valore intero.
<b>CEIL(<i>valore</i>)</b>	CEIL numerico: l'intero più piccolo maggiore o uguale a <i>valore</i> .
<b>COS(<i>valore</i>)</b>	Coseno di <i>valore</i> .
<b>COSH(<i>valore</i>)</b>	Coseno iperbolico di <i>valore</i> .
<b>EXP(<i>valore</i>)</b>	e elevato all'esponente <i>valore</i> .
<b>FLOOR(<i>valore</i>)</b>	Intero più grande minore o uguale <i>valore</i> .
<b>LN(<i>valore</i>)</b>	Logaritmo naturale di <i>valore</i> .
<b>LOG(<i>valore</i>)</b>	Logaritmo in base base 10 di <i>valore</i> .
<b>MOD(<i>valore</i>, <i>divisore</i>)</b>	Modulo.
<b>NVL(<i>valore</i>, <i>sostituto</i>)</b>	Sostituto di <i>valore</i> quando vale NULL.
<b>POWER(<i>valore</i>, <i>esponente</i>)</b>	valore elevato a POWER di un esponente.
<b>ROUND(<i>valore</i>, <i>precisione</i>)</b>	Arrotondamento di <i>valore</i> a <i>precisione</i> .
<b>SIGN(<i>valore</i>)</b>	1 se <i>valore</i> è positivo, - 1 se è negativo, 0 se è zero.
<b>SIN(<i>valore</i>)</b>	Seno di <i>valore</i> .
<b>SINH(<i>valore</i>)</b>	Seno iperbolico di <i>valore</i> .
<b>SQRT(<i>valore</i>)</b>	Radice quadrata di <i>valore</i> .
<b>TAN(<i>valore</i>)</b>	Tangente di <i>valore</i> .
<b>TANH(<i>valore</i>)</b>	Tangente iperbolica di <i>valore</i> .
<b>TRUNC(<i>valore</i>, <i>precisione</i>)</b>	Valore troncato a <i>precisione</i> .
<b>VSIZE(<i>valore</i>)</b>	Dimensione di memorizzazione di <i>valore</i> in Oracle.

## NUMERO DI VERSIONE

Il numero di versione è il numero di identificazione principale del software Oracle. In Oracle9i Release 9.0.1.0, 9 è il numero di versione.

## NUMERO SEQUENZIALE DI RIGA

È un numero assegnato a una riga al suo inserimento in un blocco dati di una tabella. Questo numero viene memorizzato insieme alla riga ed è una parte costitutiva del ROWID.

## NUMTODSINTERVAL

**VEDERE ANCHE** DATI, TIPI; Capitolo 10.

### SINTASSI

```
NUMTODSINTERVAL ( n , 'espr_car' )
```

**DESCRIZIONE** NUMTODSINTERVAL converte *n* in un letterale INTERVAL DAY TO SECOND. *n* può essere un numero oppure un'espressione che dà come risultato un numero. *espr\_car* può essere del tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2. Il valore di *espr\_car* specifica l'unità di *n* e deve dare come risultato uno di questi valori di stringa: 'GIORNO', 'ORA', 'MINUTO', o 'SECONDO'.

## NUMTOYMINTERVAL

**VEDERE ANCHE** DATI, TIPI; Capitolo 10.

### SINTASSI

```
NUMTOYMINTERVAL ( n , 'espr_car' )
```

**DESCRIZIONE** NUMTOYMINTERVAL converte il numero *n* in un letterale INTERVAL YEAR TO MONTH. *n* può essere un numero o un'espressione che dà come risultato un numero. *espr\_car* può essere del tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2. Il valore di *espr\_car* specifica l'unità di *n*, e deve dare come risultato 'ANNO' o 'MESE'.

## NUMWIDTH (SQL\*PLUS)

*Vedere* SET.

## NVARCHAR2

NVARCHAR2 è l'equivalente a più byte del tipo di dati VARCHAR2. La sua lunghezza massima è pari a 4.000 byte. *Vedere* DATI, TIPI.

## NVL

**VEDERE ANCHE** FUNZIONI DI GRUPPO, NULL, ALTRE FUNZIONI, Capitolo 8.

## SINTASSI

`NVL(valore, sostituto)`

**DESCRIZIONE** Se *valore* è NULL, questa funzione restituisce *sostituto*. Se *valore* non è NULL, questa funzione restituisce *valore*. *valore* può essere qualsiasi tipo di dati di Oracle. *sostituto* può essere un letterale, un'altra colonna o un'espressione, ma deve essere dello stesso tipo di *valore*.

## NVL2

**VEDERE ANCHE** FUNZIONI DI GRUPPO, NULL, ALTRE FUNZIONI, Capitolo 8.

## SINTASSI

`NVL2 ( espr1 , espr2 , espr3 )`

**DESCRIZIONE** NVL2 è una versione estesa di NVL. In NVL2, *espr1* non potrà mai essere restituita; al suo posto verranno restituire *espr2* o *espr3*. Se *espr1* non è NULL, NVL2 restituisce *espr2*. Se *espr1* è NULL, NVL2 restituisce *espr3*. L'argomento *espr1* può avere qualsiasi tipo di dati. Gli argomenti *espr2* e *espr3* possono avere qualunque tipo di dati tranne LONG.

## OGGETTO

Un oggetto è un elemento del database Oracle cui è possibile assegnare un nome. Gli oggetti possono dunque essere tabelle, indici, sinonimi, procedure e trigger.

## OEM

*Vedere* ORACLE ENTERPRISE MANAGER.

## OID

Un OID è un identificativo oggetto assegnato da Oracle a ciascun oggetto di un database. Per esempio, ogni oggetto riga di una tabella oggetto ha un proprio OID che viene utilizzato per risolvere i riferimenti agli oggetti riga. Oracle non riutilizza gli OID dopo che un oggetto è stato cancellato. *Vedere* il Capitolo 33.

## OPEN

**VEDERE ANCHE** CLOSE, DECLARE CURSOR, FETCH, LOOP, Capitolo 27.

## SINTASSI

`OPEN cursore [(parametro[,parametro]...)]`

**DESCRIZIONE** OPEN funziona insieme a DECLARE CURSOR e FETCH. DECLARE CURSOR configura un'istruzione SELECT da eseguire e stabilisce un elenco di parametri (variabili PL/SQL) da utilizzare nella propria clausola where, ma non esegue la query.

È OPEN cursor che esegue effettivamente la query nel cursore specificato e mantiene i risultati in un'area temporanea da dove possono essere richiamati, una riga per volta, con FETCH; i loro valori di colonna vengono inseriti in variabili locali con la clausola INTO di FETCH. Se l'istruzione SELECT del cursore ha utilizzato dei parametri, i loro valori effettivi vengono passati all'istruzione SELECT nell'elenco di parametri di OPEN. Il loro numero e la posizione devono corrispondere e devono avere tipi di dati compatibili.

Esiste anche un metodo alternativo per associare i valori di OPEN a quelli dell'elenco di SELECT:

```
DECLARE
    cursor mycur(Titolo, Editore) is select ...
BEGIN
    open my_cur(nuovo_libro = Titolo, 'PANDORAS' = Editore);
```

In questo caso, nuovo\_libro, una variabile PL/SQL caricata forse da una schermata di immissione dei dati, punta a Titolo, quindi lo carica qualsiasi sia il valore contenuto attualmente nella variabile. Editore viene caricato con il valore 'PANDORAS', e questi diventeranno i parametri del cursore *my\_cur* (per ulteriori informazioni sui parametri nei cursori, si rimanda a DECLARE CURSOR).

È inoltre possibile combinare associazioni puntate con associazioni posizionali, ma queste ultime devono apparire per prime nell'elenco di OPEN.

Non è possibile riaprire un cursore già aperto, sebbene lo si possa chiudere e riaprire. Non si può utilizzare un cursore per il quale vi sia un'istruzione OPEN pendente in un ciclo FOR.

## OPEN CURSOR (Embedded SQL)

**VEDERE ANCHE** CLOSE, DECLARE CURSOR, FETCH, PREPARE, guida di programmazione del precompilatore.

### SINTASSI

```
EXEC SQL [FOR { intero | :dimensione_array }] 
OPEN cursore
[USING
{DESCRIPTOR SQLDBA_descriptor
[:variabile[[ INDICATOR ]:variabile_indicatore
[,:variabile[ INDICATOR ]:variabile_indicatore]]...}...]
```

**DESCRIZIONE** *cursore* è il nome di un cursore dichiarato precedentemente in un'istruzione DECLARE CURSOR. Il comando USING facoltativo si riferisce all'elenco di variabili che devono essere sostituite nell'istruzione DECLARE CURSOR in base alla posizione (il numero e il tipo delle variabili devono corrispondere) oppure al nome di un descrittore che fa riferimento al risultato di un DESCRIBE precedente.

OPEN cursor alloca un cursore, definisce il gruppo attivo di righe (le variabili host vengono sostituite all'atto dell'apertura del cursore) e posiziona il cursore appena prima della riga iniziale del gruppo. Fino all'esecuzione di un FETCH non viene recuperata alcuna riga. Una volta sostituite, le variabili host non possono essere modificate. Per modificarle, si deve riaprire il cursore (in questo caso non è necessario che sia già stato chiuso).

## OPERATORE

Un operatore è un carattere o una parola riservata impiegato in un'espressione per eseguire un'operazione, come una somma o una comparazione, sugli elementi dell'espressione stessa. Alcuni esempi di operatori sono \* (moltiplicazione), > (maggiore di) e ANY (confronta un valore con tutti quelli restituiti da una subquery).

## OPERATORE RELAZIONALE

Un operatore relazionale è un simbolo utilizzato in un'espressione di ricerca per indicare un confronto tra due valori, come il segno di uguale in "where Importo = .10". Vengono restituite soltanto le righe per le quali il confronto risulta TRUE.

## OPERATORI DI QUERY

Di seguito è riportato un elenco alfabetico di tutti gli operatori di query correnti del SQL di Oracle. Ciascuno viene riportato altrove in questa guida, assieme alla sintassi e ad esempi d'uso. *Vedere anche* il Capitolo 12.

OPERATORE	SCOPO
UNION	Restituisce tutte le righe distinte da due query.
UNION ALL	Restituisce tutte le righe di una coppia di query (con eventuali ripetizioni).
INTERSECT	Restituisce tutte le righe corrispondenti da due query (senza ripetizioni).
MINUS	Restituisce tutte le righe distinte della prima query che non sono presenti nella seconda.

## OPERATORI INSIEMISTICI

Gli operatori di tipo insiemistico sono UNION, INTERSECT e MINUS. *Vedere anche* OPERATORI DI QUERY.

## OPERATORI LOGICI

**VEDERE ANCHE** PRECEDENZE.

### SINTASSI

L'elenco seguente mostra tutti gli operatori logici attualmente supportati dal linguaggio SQL di Oracle. La maggior parte di essi è trattata altrove in questo stesso glossario assieme alla propria sintassi e agli esempi d'uso. Tutti questi operatori lavorano con colonne o con letterali.

### Operatori logici che verificano un unico valore

- = *espressione* è uguale a
- > *espressione* è maggiore di
- >= *espressione* è maggiore o uguale a
- < *espressione* è minore di

<= *espressione* è minore o uguale a  
!= *espressione* è diverso da  
^= *espressione* è diverso da  
<> *espressione* è diverso da

EXISTS (*query*)  
NOT EXISTS (*query*)

LIKE *espressione*  
NOT LIKE *espressione*

*espressione* IS NULL  
*espressione* IS NOT NULL

## Operatori logici che verificano più valori

ANY (*espressione* [.*espressione*]... | *query*)  
ALL (*espressione* [.*espressione*]... | *query*)

ANY e ALL richiedono un operatore Uguale a come prefisso, per esempio >ANY, =ALL e così via.

IN (*espressione* [.*espressione*]... | *query*)  
NOT IN (*espressione* [.*espressione*]... | *query*)

BETWEEN *espressione* AND *espressione*  
NOT BETWEEN *espressione* AND *espressione*

## Altri operatori logici

+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione
**	Elevamento a potenza
	Concatenazione
()	Ignora le normali regole di precedenza, o racchiude una subquery
NOT	Inverte l'espressione logica
AND	Combina espressioni logiche
OR	Combina espressioni logiche
UNION	Combina i risultati delle query
UNION ALL	Combina i risultati delle espressioni senza eliminare i duplicati INTERSECT
INTERSECT	Combina i risultati delle query
MINUS	Combina i risultati delle query

## OPERATORI SINTATTICI

**VEDERE ANCHE** OPERATORI LOGICI, PRECEDENZE.

**DESCRIZIONE** Gli operatori sintattici hanno la precedenza più alta tra tutti gli operatori, e possono comparire ovunque all'interno di istruzioni SQL. Di seguito sono elencati in ordine decrescente di precedenza. Gli operatori con la stessa precedenza vengono valutati da sinistra a destra. La maggior parte di questi operatori è elencata e descritta separatamente in questo stesso capitolo.

OPERATORE	FUNZIONE
-	Continuazione di un comando SQL*PLUS. Continua un comando sulla linea seguente.
&	Prefisso dei parametri di un file di avvio di SQL*PLUS. Al posto di &1, &2, ecc. vengono sostituiti i parametri della linea di comando. Vedere START.
& &&	Prefisso di una variabile di sostituzione di un comando SQL in SQL*PLUS. SQL*PLUS richiede l'inserimento di un valore se trova una variabile & o && non definita. && ha l'effetto di definire la variabile e salva il valore; '&' invece non lo fa. Vedere & e &&, DEFINE e ACCEPT.
:	Prefisso di una variabile di SQL*FORMS e di una variabile host di PL/SQL.
.	Separatore di variabili, utilizzato in SQL*PLUS per separare il nome di variabile da un suffisso in modo che questo non sia considerato parte del nome stesso. In SQL separa nomi utente, nomi di tabella e nomi di colonna.
( )	Racchiudono subquery ed elenchi di colonne o amministrano le precedenze.
'	Racchiudono letterali, come stringhe di caratteri o date. Per includere un apice in una stringa, occorre usarne due (non le virgolette).
"	Racchiudono l'alias di una tabella o di una colonna che contiene caratteri speciali o spazi.
"	Racchiudono il testo letterale nella clausola di un formato di data di TO_CHAR.
@	Precede il nome di un database nel comando COPY o il nome di un collegamento nella clausola from.

## OPSS LOGIN

Gli OPSS LOGIN sono particolari nomi utente di Oracle, in cui OPSS è anteposto all'ID di account del sistema operativo dell'utente per semplificare l'accesso a Oracle da questo ID.

## OR

**VEDERE ANCHE** AND, CONTAINS, Capitoli 3 e 24.

**DESCRIZIONE** OR combina espressioni logiche in modo che il risultato sia vero se almeno una delle espressioni di partenza è vera.

### ESEMPIO

L'esempio seguente produce dati sia per KEENE sia per SAN FRANCISCO:

```
select * from COMFORT
where Citta = 'KEENE' OR Citta = 'SAN FRANCISCO'
```

All'interno di query CONTEXT, l'operatore OR può essere utilizzato per ricerche che riguardano più termini di ricerca. Nel caso in cui uno dei termini cercati venga trovato e il suo punteggio superi il valore di soglia specificato, il testo viene restituito dalla ricerca. Vedere il Capitolo 24.

## ORACLE ENTERPRISE MANAGER

Oracle Enterprise Manager (OEM) è un prodotto della Oracle Corporation. OEM mette a disposizione un'interfaccia grafica per attività di DBA comuni e dispone di caratteristiche aggiuntive per le capacità di gestione dei dati e di messa a punto delle prestazioni estese.

## ORACLE REAL APPLICATION CLUSTER

In un ambiente Real Application Cluster (RAC) di Oracle, più istanze (solitamente su un cluster di server) accedono a un unico gruppo di file di dati. RAC fornisce la capacità di failover del server in un ambiente ad alta disponibilità, dato che gli utenti possono essere indirizzati a più server per accedere agli stessi dati.

## ORACLE9i APPLICATION SERVER

Oracle9i Application Server (9iAS) consente agli sviluppatori di accedere a database Oracle da applicazioni basate sul Web. Gli sviluppatori chiamano le procedure dall'interno del database. Le procedure, a loro volta, recuperano o manipolano i dati e restituiscono i risultati agli sviluppatori. Vedere il Capitolo 39.

## ORDBMS

*Vedere RELAZIONALE A OGGETTI, SISTEMA DI GESTIONE DI DATABASE.*

## ORDER BY

**VEDERE ANCHE** COLLATION, FROM, GROUP BY, HAVING, SELECT, WHERE.

### SINTASSI

```
ORDER BY { espressione [.espressione]... |  
          posizione [.posizione]... }  
          alias [.alias]... }  
          [ ASC | DESC ]
```

**DESCRIZIONE** La clausola ORDER BY fa sì che Oracle ordini i risultati di una query prima della loro visualizzazione. Questo risultato può essere ottenuto tramite le *espressioni*, che possono essere semplici nomi di colonna o insiemi complessi di funzioni, attraverso *alias* (*vedere* la nota seguente) o mediante la *posizione* di una colonna nella clausola select (*vedere* la nota seguente). Le righe vengono ordinate in base alla prima espressione o posizione, poi in base alla seconda e così via, seguendo la sequenza di ordinamento dell'host. Se non si specifica ORDER BY, l'ordine con cui le righe vengono selezionate da una tabella è indeterminato e può cambiare da una query all'altra.

**NOTA** Oracle supporta ancora l'uso delle posizioni di colonna nelle clausole ORDER BY, ma questa caratteristica non fa più parte dello standard SQL e non è sicuro che verrà supportata nelle release future. Al suo posto si utilizzano gli alias di colonna.

ASC e DESC specificano se l'ordinamento dev'essere crescente o decrescente e possono seguire qualsiasi espressione, posizione o alias in ORDER BY. In Oracle, i valori NULL precedono le righe crescenti e seguono quelle decrescenti.

ORDER BY segue qualsiasi altra clausola tranne FOR UPDATE OF.

Se ORDER BY e DISTINCT vengono specificati insieme, la clausola ORDER BY può riferirsi solo alle colonne e alle espressioni contenute nella clausola SELECT.

Quando si usano gli operatori UNION, INTERSECT o MINUS, i nomi delle colonne nella prima selezione possono differire da quelli delle selezioni successive. ORDER BY deve usare il nome di colonna preso dalla prima selezione.

In un ORDER BY possono apparire solo tipi di dati CHAR, VARCHAR2, NUMBER, DATE e il tipo speciale RowID.

### ESEMPIO

```
select Titolo, Editore
  from BIBLIOTECA
 order by Titolo;
```

## ORDINAMENTO

**VEDERE ANCHE** GROUP BY, INDEX, ORDER BY, Capitolo 9.

**DESCRIZIONE** La sequenza di ordinamento è l'ordine in cui caratteri, numeri e simboli vengono ordinati a causa di una clausola order by o group by. Queste sequenze differiscono in relazione alla sequenza di confronto del sistema operativo del computer o alla lingua della nazione. Le sequenze EBCDIC (utilizzate solitamente per mainframe IBM e compatibili) e ASCII (utilizzate nella maggior parte dei computer) differiscono tra loro in modo significativo. Nonostante queste differenze, è sempre possibile applicare le seguenti regole:

- un numero con un valore più grande viene considerato “maggiori” di quello con il valore più piccolo; tutti i numeri negativi vengono considerati minori di quelli positivi; perciò –10 è minore di 10; –100 è minore di –10.
- Una data successiva è considerata maggiore di una precedente.

Le stringhe di caratteri vengono confrontate lettera per lettera a partire da quella più a sinistra fino al primo carattere differente. La stringa che possiede il carattere “maggiori” in quella posizione viene considerata la più grande. Un carattere è considerato più grande di un altro quando ha un codice ASCII (o EBCDIC) più elevato. Questo significa generalmente che B è maggiore di A ma anche che il valore di A in rapporto ad “a” o a “1” varia da piattaforma a piattaforma.

Queste sequenze di confronto variano leggermente a seconda se si stanno utilizzando stringhe CHAR o VARCHAR2.

Se due stringhe VARCHAR2 sono identiche fino alla fine di quella più corta, la più lunga viene considerata maggiore. Se due stringhe sono identiche e per di più hanno la stessa lunghezza sono considerate uguali.

Con stringhe di tipo CHAR, la stringa più breve viene riempita con spazi fino alla lunghezza della stringa più lunga. Se dopo questa operazione le stringhe sono diverse, il confronto tratta gli spazi aggiunti come minori di qualsiasi altro carattere, dando come risultato lo stesso valore del confronto di tipo VARCHAR2. Se, dopo aver aggiunto gli spazi, ma non prima, le stringhe risultano identiche, il confronto di tipo CHAR le tratta come uguali, a differenza di quanto fa il confronto di tipo VARCHAR2.

In SQL è importante che i numeri letterali vengano scritti senza essere racchiusi da apici, in quanto ‘10’ verrebbe considerato minore di ‘6’, poiché gli apici fanno in modo che vengano considerati come stringhe di caratteri anziché numeri e ‘6’ viene considerato maggiore di ‘1’, che si trova alla prima posizione di ‘10’.

## OTTIMIZZATORE

L'ottimizzatore è quella parte del kernel di Oracle che sceglie il modo migliore di utilizzare tabelle e indici per completare la richiesta di un'istruzione SQL. Vedere il Capitolo 38.

## OUTER JOIN

Vedere JOIN per una trattazione dettagliata. A partire da Oracle9i, si potranno utilizzare le opzioni per la sintassi degli outer join dello standard ANSI (LEFT OUTER JOIN, RIGHT OUTER JOIN e FULL OUTER JOIN). Per dettagli e esempi, si rimanda al Capitolo 12.

## PACKAGE

Un package è un oggetto PL/SQL che raggruppa tipi di dati PL/SQL, variabili, cursori SQL, eccezioni, procedure e funzioni. Ogni package possiede una specifica e un corpo. La specifica mostra gli oggetti cui si può accedere mentre si utilizza il package. Il corpo definisce completamente tutti gli oggetti e può contenerne altri utilizzati solo per i processi interni. È possibile modificare il corpo (per esempio aggiungendo nuove procedure al package) senza invalidare gli oggetti utilizzati dal package.

Vedere CREATE PACKAGE, CREATE PACKAGE BODY, CURSOR, ECCEZIONE, FUNZIONE, TABELLA (PL/SQL), RECORD (PL/SQL), PROCEDURA e il Capitolo 29.

## PADRE

In un albero, un padre è un nodo che possiede un altro nodo sotto di lui (un figlio).

## PAGESIZE (SQL\*PLUS)

Vedere SET.

## PARALLEL QUERY OPTION

Parallel Query Option (PQO) suddivide un'unica attività del database in più attività coordinate, consentendo all'attività di utilizzare più processori. Per esempio, una scansione intera di tabella oppure un'operazione di ordinamento di grosse dimensioni possono essere eseguite in parallelo, permettendo a più processori di partecipare al completamento dell'operazione. Se sul server sono disponibili risorse adeguate, l'operazione eseguita in parallelo potrà essere completata più velocemente di quanto sarebbe possibile se venisse eseguita come un'attività singola.

Il numero di processi server di query parallele utilizzati per eseguire un'operazione prende il nome di *grado di parallelismo*, e viene impostato tramite il parametro DEGREE dei suggerimenti. Vedere il Capitolo 38.

## PARAMETRI

**VEDERE ANCHE** &, &&, ACCEPT, DEFINE, Capitolo 16.

**DESCRIZIONE** I parametri permettono di eseguire un file di avvio con valori passati dalla linea di comando. Questi vengono inseriti semplicemente dopo il nome del file di avvio separati gli uni dagli altri da uno spazio. All'interno del file, essi sono indicati in base all'ordine in cui apparivano sulla linea di comando. &1 è il primo, &2 il secondo e così via. Le regole d'uso sono le stesse delle variabili caricate con DEFINE o ACCEPT e possono essere utilizzate allo stesso modo anche nelle istruzioni SQL.

Esiste comunque una limitazione. Non esiste un modo per passare un argomento composto da più parole a una variabile singola. Ciascuna variabile può accettare una sola parola, data o numero. Se però si racchiudono i parametri tra apici sulla linea di comando, le parole vengono concatenate senza particolari problemi.

### **ESEMPIO**

Si supponga di avere un file di avvio di nome fred.sql che contiene il codice SQL seguente:

```
select Titolo, Editore from BIBLIOTECA
where Titolo &1;
```

Se lo si esegue attraverso il comando seguente:

```
start fred.sql M
```

produce un report di tutti i titoli che iniziano con un carattere ‘M’.

### **PARAMETRO**

Un parametro è un valore, il nome di una colonna o un’espressione che in genere segue una funzione o il nome di un modulo. Serve a specificare funzioni o controlli aggiuntivi che devono essere rispettati dalla funzione o dal modulo chiamato. Per un esempio, *vedere PARAMETRI*.

### **PARENTESI**

*Vedere OPERATORI LOGICI e PAROLE CHIAVE E SIMBOLI PL/SQL.*

### **PAROLE RISERVATE**

#### **VEDERE ANCHE NOMI DEGLI OGGETTI.**

**DESCRIZIONE** Una parola riservata ha un significato particolare per SQL e per questo motivo non può essere utilizzata come nome di un oggetto. Questo non vale per alcune parole chiave che possono essere utilizzate come nomi di oggetti, ma in seguito possono diventare parole riservate. Non tutte le parole sono riservate in tutti i prodotti, pertanto l’elenco seguente cerca di visualizzare quali prodotti riservano le varie parole. Nella tabella seguente, le voci con gli asterischi sono riservate sia in SQL sia in PL/SQL. Le voci senza asterisco sono riservate solo in PL/SQL.

ACCESS	ALL*	ALTER*	AND*	ANY*
ARRAY	AS*	ASC*	AT	AUDIT
AUTHID	AVG	BEGIN	BETWEEN*	BINARY_INTEGER
BODY	BOOLEAN	BULK	BY*	CASE
CHAR*	CHAR_BASE	CHECK*	CLOSE	CLUSTER*
COALESCE	COLLECT	COLUMN	COMMENT*	COMMIT
COMPRESS*	CONNECT*	CONSTANT	CREATE*	CURRENT*

(segue)

(continua)

CURRVAL	CURSOR	DATE*	DAY	DECLARE
DECIMAL*	DEFAULT*	DELETE*	DESC*	DISTINCT*
DO	DROP*	ELSE*	ELSIF	END
EXCEPTION	EXCLUSIVE*	EXECUTE	EXISTS*	EXIT
EXTENDS	EXTRACT	FALSE	FETCH	FILE
FLOAT*	FOR*	FORALL	FROM*	FUNCTION
GOTO	GROUP*	HAVING*	HEAP	HOUR
IDENTIFIED	IF	IMMEDIATE*	IN*	INCREMENT
INDEX*	INDICATOR	INITIAL	INSERT*	INTEGER*
INTERFACE	INTERSECT*	INTERVAL	INTO*	IS*
ISOLATION	JAVA	LEVEL*	LIKE*	LIMITED
LOCK*	LONG*	LOOP	MAX	MAXEXTENTS
MIN	MINUS*	MINUTE	MLSLABEL*	MOD
MODE*	MODIFY	MONTH	NATURAL	NATURALN
NEW	NEXTVAL	NOCOMPRESS	NOCOPY	NOT*
NOWAIT*	NULL*	NULLIF	NUMBER*	NUMBER_BASE
OCIROWID	OF*	OFFLINE	ON*	ONLINE
OPAQUE	OPEN	OPERATOR	OPTION*	OR*
ORDER*	ORGANIZATION	OTHERS	OUT	PACKAGE
PARTITION	PCTFREE*	PLS_INTEGER	POSITIVE	POSITIVEN
PRAGMA	PRIOR*	PRIVATE	PROCEDURE	PUBLIC*
RAISE	RANGE	RAW*	REAL	RECORD
REF	RELEASE	RENAME	RESOURCE	RETURN
REVERSE	ROLLBACK	ROW*	ROWID*	ROWNUM*
ROWTYPE	SAVEPOINT	SECOND	SELECT*	SEPARATE
SET*	SHARE*	SMALLINT*	SPACE	SQL
SQLCODE	SQLERRM	START*	STDDEV	SUBTYPE
SUCCESSFUL*	SUM	SYNONYM*	SYSDATE*	TABLE*
THEN*	TIME	TIMESTAMP	TIMEZONE_REGION	TIMEZONE_ABBR

(segue)

*(continua)*

TIMEZONE_MINUTE	TIMEZONE_HOUR	TO*	TRIGGER*	TRUE
TYPE	UI	UID	UNION	UNIQUE
UPDATE	USER	VALIDATE	VALUES	VARCHAR
VARCHAR2	VIEW	WHENEVER	WHERE	WITH

## PARTIZIONARE

Partizionare una tabella significa dividere sistematicamente le sue righe tra più tabelle, ciascuna con la stessa struttura. È possibile imporre al database di partizionare automaticamente una tabella e i relativi indici. *Vedere* il Capitolo 18.

## PARTIZIONE COMPOSTA

Una partizione composta implica l'uso di più metodi di partizione, come una tabella partizionata in base agli intervalli in cui le partizioni basate sugli intervalli vengono sottoposte all'hash partitioning. *Vedere* il Capitolo 18.

## PARTIZIONE HASH

In una partizione hash, i dati da partizionare vengono elaborati da un algoritmo di hash invece che essere semplicemente partizionati per intervallo. L'hash partitioning suddivide piccoli gruppi di dati sequenziali tra più partizioni rispetto al partizionamento a intervalli. *Vedere* CREATE TABLE.

## PARTIZIONE SECONDARIA

Una partizione secondaria è una partizione di una partizione. Solitamente, una partizione a intervalli viene partizionata con hash partitioning, creando così più partizioni secondarie hash per ciascuna delle partizioni a intervalli. *Vedere* il Capitolo 18.

## PASSWORD

Una password è una stringa di caratteri che si deve inserire quando ci si connette al sistema operativo del computer host o a un database Oracle. Le password dovrebbero rimanere segrete.

## PASSWORD (comando SQL\*PLUS)

**VEDERE ANCHE** ALTER USER, Capitolo 19.

### SINTASSI

PASSW[ORD] [nomeutente]

**DESCRIZIONE** Il comando password può essere utilizzato in SQL\*Plus per modificare la password nel proprio account o in quello di un altro utente (se si dispone del privilegio ALTER ANY USER). Se si usa questo comando, la nuova password non compare sullo schermo mentre la si digita. Gli utenti con autorità DBA possono modificare la password di un altro utente mediante il comando password; gli altri utenti possono modificare solo la propria password.

Quando si impedisce il comando password, il sistema chiede di inserire prima la vecchia password e poi quella nuova.

### ESEMPIO

```
password  
Changing password for dora  
Old password:  
New password:  
Retype new password:
```

Una volta che la password è stata cambiata con successo il sistema avvisa con il messaggio:

```
Password changed
```

### PAUSE (Forma 1, SQL\*PLUS)

Vedere SET.

### PAUSE (Forma 2, SQL\*PLUS)

**VEDERE ANCHE** ACCEPT, PROMPT, Capitolo 6.

#### SINTASSI

```
PAU[SE] [testo];
```

**DESCRIZIONE** PAUSE è simile a PROMPT, ma visualizza prima una linea vuota, poi una linea contenente *testo*, quindi attende che l'utente prema INVIO. Se non si è inserito alcun *testo*, PAUSE visualizza due linee vuote e attende la pressione di INVIO.

**NOTA** PAUSE attende la pressione di INVIO dal terminale anche quando la sorgente dei comandi è stata dirottata da un file. In altre parole, un file di avvio contenente SET TERMOUT OFF potrebbe arrestarsi, attendere la pressione di INVIO senza fornire alcun messaggio del perché è in attesa (o per informare l'utente che è in attesa). Si usi questo comando con molta cautela.

### ESEMPIO

```
prompt Report completo.  
pause Premere INVIO per continuare.
```

### PCTFREE

PCTFREE è una porzione di un blocco di dati che non viene riempita dalle righe inserite in una tabella, ma è riservata ad aggiornamenti successivi sulle righe del blocco stesso.

## PCTUSED

PCTUSED è la percentuale di spazio in un blocco di dati che Oracle cerca di mantenere pieno. Se la percentuale impiegata è inferiore al valore di PCTUSED, allora il blocco viene aggiunto all'elenco dei blocchi liberi nel segmento. PCTUSED può essere impostato su una base tabella per tabella.

## PERCENT\_RANK

**VEDERE ANCHE** FUNZIONI DI GRUPPO, Capitolo 13.

### SINTASSI

Per gli aggregati:

```
PERCENT_RANK ( expr [, expr]... ) WITHIN GROUP
( ORDER BY
  expr [ DESC | ASC ] [NULLS { FIRST | LAST }]
  [, expr [ DESC | ASC ] [NULLS { FIRST | LAST }]]...)
```

Per le funzioni analitiche:

```
PERCENT_RANK ( ) OVER ( [clausola_partizionamento_query] clausola_order_by )
```

**DESCRIZIONE** In qualità di funzione di aggregazione, PERCENT\_RANK calcola, per un'ipotetica riga R identificata dagli argomenti della funzione e da una corrispondente specifica di ordinamento, il grado della riga R meno 1 diviso per il numero di righe presenti nel gruppo di aggregazione. Questo calcolo viene eseguito come se l'ipotetica riga R si trovasse nel gruppo di righe sul quale Oracle esegue l'operazione di aggregazione. Gli argomenti della funzione identificano una singola riga ipotetica all'interno di ciascun gruppo di aggregazione. Di conseguenza, devono essere valutati tutti a espressioni costanti all'interno di ciascun gruppo di aggregazione. Le espressioni dell'argomento costanti e le espressioni nella clausola ORDER BY dell'aggregato corrispondono per posizione. Quindi, il numero di argomenti dev'essere uguale e i relativi tipi di dati devono essere compatibili.

L'intervallo dei valori restituiti da PERCENT\_RANK va da 0 a 1 compresi. La prima riga in qualsiasi gruppo ha un valore di PERCENT\_RANK pari a 0.

In qualità di funzione analitica, PERCENT\_RANK calcola per una riga R il grado di R meno 1, diviso per il numero di righe valutate (l'intero gruppo dei risultati di una query oppure una partizione) meno 1.

Per un esempio sull'impiego di PERCENT\_RANK si rimanda al Capitolo 13.

## PERCENTILE\_CONT

**VEDERE ANCHE** FUNZIONI DI GRUPPO, PERCENTILE\_DISC, Capitolo 13.

### SINTASSI

```
PERCENTILE_CONT ( expr ) WITHIN GROUP ( ORDER BY expr [ DESC | ASC ] )
[OVER ( clausola_partizionamento_query )]
```

**DESCRIZIONE** PERCENTILE\_CONT è una funzione di distribuzione inversa che presuppone l'uso di un modello a distribuzione continua. Accetta un valore percentile e una specifica di

ordinamento e restituisce un valore interpolato che rientrerebbe in questo valore percentile rispetto alla specifica di ordinamento. Nel calcolo i valori NULL vengono ignorati.

La prima *espr* deve essere valutata a un valore numerico compreso tra 0 e 1, poiché si tratta di un valore percentile. Questa *espr* deve rimanere costante all'interno di ciascun gruppo di aggregazione. La clausola ORDER BY accetta una singola espressione che deve essere un valore numerico e un valore di data/ora, poiché questi sono i tipi per i quali Oracle può eseguire un'interpolazione.

## **PERCENTILE\_DISC**

**VEDERE ANCHE** FUNZIONI DI GRUPPO, PERCENTILE\_CONT, Capitolo 13.

### **SINTASSI**

```
PERCENTILE_DISC ( espr ) WITHIN GROUP ( ORDER BY espr [ DESC | ASC ] )
[OVER ( clausola_partizionamento_query )]
```

**DESCRIZIONE** PERCENTILE\_DISC è una funzione di distribuzione inversa che presuppone l'uso di un modello a distribuzione discreta. Accetta un valore percentile e una specifica di ordinamento e restituisce un elemento dal gruppo. Nel calcolo i valori NULL vengono ignorati.

La prima *espr* dev'essere valutata a un valore numerico compreso tra 0 e 1, poiché si tratta di un valore percentile. Questa espressione deve rimanere costante all'interno di ciascun gruppo di aggregazione. La clausola ORDER BY accetta una singola espressione che può essere di qualunque tipo ordinabile.

Per un determinato valore percentile P, la funzione PERCENTILE\_DISC ordina i valori dell'espressione nella clausola ORDER BY, e restituisce quello con il valore CUME\_DIST più piccolo (rispetto alla medesima specifica di ordinamento) che è maggiore o uguale a P.

## **POWER**

**VEDERE ANCHE** SQRT, Capitolo 8.

### **SINTASSI**

```
POWER(valore,esponente)
```

**DESCRIZIONE** POWER è *valore* elevato a un *esponente*.

### **ESEMPIO**

```
POWER(2,4) = 16
```

## **PRAGMA**

Un'istruzione pragma è una direttiva del compilatore, piuttosto che una porzione di codice eseguibile. Nonostante assomigli al codice eseguibile e appaia nei listati dei programmi (come un blocco PL/SQL), un'istruzione pragma non è effettivamente eseguibile, e per questo motivo non appare nel codice di esecuzione del blocco. Il suo compito è quello di fornire istruzioni al compilatore. Vedere EXCEPTION\_INIT e il Capitolo 30.

## PRECEDENZE

**VEDERE ANCHE** OPERATORI LOGICI, OPERATORI DI QUERY, Capitolo 12.

**DESCRIZIONE** Gli operatori seguenti sono elencati in ordine decrescente di precedenza. Gli operatori con uguale precedenza sono posti sulla stessa linea. Inoltre, vengono valutati in successione da sinistra a destra. Tutti gli AND sono valutati prima di qualsiasi OR. Questi operatori sono elencati e descritti separatamente al proprio simbolo o nome in questo stesso capitolo.

OPERATORE	FUNZIONE
-	Continuazione di un comando SQL*PLUS. Continua un comando sulla linea seguente.
&	Prefisso dei parametri di un file di avvio di SQL*PLUS. &1, &2, ecc. vengono sostituiti da parole. Vedere START.
& &&	Prefisso di sostituzione di un comando SQL in SQL*PLUS. SQL*PLUS richiede l'inserimento di un valore se trova una variabile & o && non definita. && ha l'effetto di definire la variabile e salva il valore; '&' invece non lo fa. Vedere & e &&, DEFINE e ACCEPT.
:	Prefisso di una variabile host di PL/SQL.
.	Separatore di variabili, utilizzato in SQL*PLUS per separare il nome di una variabile da un suffisso in modo che quest'ultimo non venga considerato parte del nome di variabile.
()	Racchiudono le subquery o gli elenchi di colonne.
'	Racchiudono letterali, come stringhe di caratteri o costanti delle date. Per includere un apice in una costante di stringa se ne usino due (non si usino le virgolette).
"	Racchiudono l'alias di una tabella o di una colonna nel caso contenga caratteri speciali o spazi.
"	Racchiudono il testo letterale nella clausola di un formato di data di TO_CHAR.
@@	Precede il nome di un database nel comando COPY o il nome di un collegamento nella clausola from.
( )	Sovrascrivono la normale precedenza degli operatori.
+ -	Segno di prefisso (positivo o negativo) di un numero o di un'espressione numerica.
* /	Moltiplicazione e divisione.
+ -	Addizione e sottrazione.
	Concatenazione di caratteri.
NOT	Inverte il risultato di un'espressione.
AND	È vera se entrambe le condizioni sono vere.
OR	È vera se almeno una condizione è vera.
UNION	Restituisce tutte le righe separate risultanti da due query.
INTERSECT	Restituisce tutte le righe separate identiche da due query.
MINUS	Restituisce tutte le righe separate della prima query che non abbiano corrispondenza nella seconda.

## PRECOMPILATORE

Il precompilatore è un programma che legge un codice sorgente particolare e scrive un file del programma sorgente modificato (precompilato) pronto per essere letto da un compilatore normale.

## PREDICATO

Un predicato è la clausola where, e più esplicitamente un criterio di selezione basato su uno degli operatori relazionali (=, !=, IS, IS NOT, >, >=) e non contenente alcun operatore logico (AND, OR o NOT).

## PREPARE (Embedded SQL)

**VEDERE ANCHE** CLOSE, DECLARE, CURSOR, FETCH, OPEN, guida di programmazione del precompilatore.

### SINTASSI

```
EXEC SQL [AT { db_nome | :variabile_host }] PREPARE istruzione_id  
FROM { :stringa_host | "testo" | comando_select }
```

**DESCRIZIONE** PREPARE analizza SQL all'interno della variabile host *:stringa\_host* o del *testo* letterale. Inoltre assegna un *istruzione\_id* come riferimento a SQL. Se *istruzione\_id* è già stato utilizzato precedentemente, questo riferimento lo sovrascrive. Quando l'istruzione fornita è *select* vi si può includere una clausola *for update of*. *:stringa\_host* non è il vero nome della variabile host impiegata, ma un segnaposto. OPEN CURSOR assegna le variabili host di input nella propria clausola using, mentre FETCH assegna le variabili host di output nella propria clausola into, in base alla posizione. Un'istruzione deve essere preparata una sola volta. Quindi, potrà essere eseguita più volte.

### ESEMPIO

```
stringa_query : string(1..100)  
get(stringa_query);  
EXEC SQL prepare FRED from :stringa_query;  
EXEC SQL execute FRED;
```

## PRINT

**VEDERE ANCHE** BIND VARIABLE, VARIABLE, Capitolo 26.

### SINTASSI

```
PRINT variabile1 variabile2
```

**DESCRIZIONE** Visualizza il valore corrente della variabile specificata (creata con il comando VARIABLE). È possibile visualizzare i valori correnti di più variabili in un unico comando.

## PRIOR

Vedere CONNECT BY.

## PRIVILEGI

Un privilegio è un permesso concesso a un utente Oracle per l'esecuzione di alcune azioni. In Oracle, nessun utente può eseguire nessuna azione senza aver prima ottenuto il privilegio opportuno.

Esistono due tipologie di privilegi: i privilegi di sistema e i privilegi sugli oggetti. I privilegi di sistema sono permessi per l'esecuzione di comandi di definizione e controllo dei dati, come CREATE TABLE o ALTER USER; è un privilegio di sistema persino quello che permette di collegarsi al database. I privilegi sugli oggetti consentono di operare su particolari oggetti del database.

I privilegi di sistema di Oracle comprendono CREATE, ALTER e DROP che permettono di impartire i vari comandi CREATE, ALTER e DROP. I privilegi che contengono la parola chiave ANY consentono all'utente di esercitare il privilegio su qualsiasi schema per il quale questo privilegio è stato concesso, e non solo sugli schemi di sua proprietà. I privilegi standard, riportati nell'elenco seguente, forniscono semplicemente il permesso di eseguire il comando indicato e non richiedono ulteriori spiegazioni. Altri, invece, non sono così ovvi, quindi vengono spiegati brevemente.

I privilegi di sistema includono:

PRIVILEGIO	PERMESSO RELATIVO
<b>CLUSTER</b>	
CREATE CLUSTER	Creare dei cluster nello schema del concessionario.
CREATE ANY CLUSTER	Creare un cluster in qualsiasi schema. Si comporta analogamente a CREATE ANY TABLE.
ALTER ANY CLUSTER	Modificare cluster in qualunque schema.
DROP ANY CLUSTER	Cancellare cluster in qualsiasi schema.
<b>CONTEXT</b>	
CREATE ANY CONTEXT	Creare qualsiasi spazio di nomi contestuale.
DROP ANY CONTEXT	Cancellare qualsiasi spazio di nomi contestuale.
<b>DATABASE</b>	
ALTER DATABASE	Modificare il database.
ALTER SYSTEM	Eseguire le istruzioni ALTER SYSTEM.
AUDIT SYSTEM	Eseguire istruzioni <i>istruzioni_sql</i> di AUDIT.
<b>DATABASE LINK</b>	
CREATE DATABASE LINK	Creare database link privati nello schema del concessionario.
CREATE PUBLIC DATABASE LINK	Creare database link pubblici.
DROP PUBLIC DATABASE LINK	Cancellare database link pubblici.
<b>DIMENSION</b>	
CREATE DIMENSION	Creare dimensioni nello schema del concessionario.

(segue)

(continua)

<b>PRIVILEGIO</b>	<b>PERMESSO RELATIVO</b>
CREATE ANY DIMENSION	Creare dimensioni in qualsiasi schema.
ALTER ANY DIMENSION	Modificare le dimensioni in qualunque schema.
DROP ANY DIMENSION	Cancellare le dimensioni in qualsiasi schema.
<b>DIRECTORY</b>	
CREATE ANY DIRECTORY	Creare oggetti di database della directory.
DROP ANY DIRECTORY	Cancellare oggetti di database della directory.
<b>INDEXTYPE</b>	
CREATE INDEXTYPE	Creare un tipo di indice nello schema del concessionario.
CREATE ANY INDEXTYPE	Creare un tipo di indice in qualsiasi schema.
ALTER ANY INDEXTYPE	Modificare i tipi di indici in qualunque schema.
DROP ANY INDEXTYPE	Eliminare un tipo di indice in qualunque schema.
EXECUTE ANY INDEXTYPE	Fare riferimento a un tipo di indice in qualsiasi schema.
<b>INDEX</b>	
CREATE ANY INDEX	Creare un indice di dominio in qualsiasi schema oppure un indice su qualsiasi tabella di qualunque schema.
ALTER ANY INDEX	Modificare gli indici in qualsiasi schema.
DROP ANY INDEX	Eliminare gli indici in qualunque schema.
<b>LIBRARY</b>	
CREATE LIBRARY	Creare librerie di procedure/funzioni esterne nello schema del concessionario.
CREATE ANY LIBRARY	Creare librerie di procedure/funzioni esterne in qualsiasi schema.
DROP ANY LIBRARY	Eliminare librerie di procedure/funzioni esterne in qualsiasi schema.
<b>MATERIALIZED VIEW</b>	
CREATE MATERIALIZED VIEW	Creare una vista materializzata nello schema del concessionario.
CREATE ANY MATERIALIZED VIEW	Creare viste materializzate in qualunque schema.
ALTER ANY MATERIALIZED VIEW	Modificare viste materializzate in qualsiasi schema.
DROP ANY MATERIALIZED VIEW	Eliminare viste materializzate in qualsiasi schema.
QUERY REWRITE	Abilitare la riscrittura tramite una vista materializzata, oppure creare un indice basato su una funzione quando questa vista materializzata o questo indice fanno riferimento a tabelle e viste che si trovano nello schema del concessionario.

(segue)

*(continua)*

<b>PRIVILEGIO</b>	<b>PERMESSO RELATIVO</b>
GLOBAL QUERY REWRITE	Abilitare la riscrittura tramite una vista materializzata, oppure creare un indice basato su una funzione quando questa vista materializzata o questo indice fanno riferimento a tabelle e viste di qualsiasi schema.
ON COMMIT REFRESH	Creare una vista materializzata di tipo ad "aggiornamento su commit" su qualsiasi tabella del database. Modificare una vista materializzata di tipo ad "aggiornamento su richiesta" su qualsiasi tabella nel database da aggiornare su commit.
<b>OPERATOR</b>	
CREATE OPERATOR	Creare un operatore e le relative associazioni nello schema del concessionario.
CREATE ANY OPERATOR	Creare un operatore e le relative associazioni in qualunque schema.
DROP ANY OPERATOR	Eliminare un operatore in qualunque schema.
EXECUTE ANY OPERATOR	Fare riferimento a un operatore in qualsiasi schema.
<b>OUTLINE</b>	
CREATE ANY OUTLINE	Creare strutture pubbliche che possono essere utilizzate in qualsiasi schema se serve di strutture.
ALTER ANY OUTLINE	Modificare le strutture.
DROP ANY OUTLINE	Eliminare le strutture.
SELECT ANY OUTLINE	Creare una struttura privata che sia il clone di una struttura pubblica.
<b>PROCEDURE</b>	
CREATE PROCEDURE	Creare stored procedure, funzioni e package nello schema del concessionario.
CREATE ANY PROCEDURE	Creare stored procedure, funzioni e package in qualunque schema.
ALTER ANY PROCEDURE	Modificare stored procedure, funzioni o package in qualsiasi schema.
DROP ANY PROCEDURE	Eliminare stored procedure, funzioni o package in qualunque schema.
EXECUTE ANY PROCEDURE	Eseguire procedure o funzioni (indipendenti o raccolte in package). Fare riferimento a variabili di package pubblici in qualunque schema.
<b>PROFILE</b>	
CREATE PROFILE	Creare profili.
ALTER PROFILE	Modificare profili.
DROP PROFILE	Eliminare profili.
<b>ROLE</b>	
CREATE ROLE	Creare ruoli.
ALTER ANY ROLE	Modificare qualsiasi ruolo nel database.

*(segue)*

(continua)

<b>PRIVILEGIO</b>	<b>PERMESSO RELATIVO</b>
DROP ANY ROLE	Eliminare ruoli.
GRANT ANY ROLE	Concedere qualunque ruolo nel database.
<b>ROLLBACK SEGMENT</b>	
CREATE ROLLBACK SEGMENT	Creare segmenti di rollback.
ALTER ROLLBACK SEGMENT	Modificare segmenti di rollback.
DROP ROLLBACK SEGMENT	Eliminare segmenti di rollback.
<b>SEQUENCES</b>	
CREATE SEQUENCE	Creare sequenze nello schema del concessionario.
CREATE ANY SEQUENCE	Creare sequenze in qualsiasi schema.
ALTER ANY SEQUENCE	Modificare qualunque sequenze nel database.
DROP ANY SEQUENCE	Eliminare sequenze in qualsiasi schema.
SELECT ANY SEQUENCE	Fare riferimento a sequenze in qualunque schema.
<b>SESSION</b>	
CREATE SESSION	Connetersi al database.
ALTER RESOURCE COST	Impostare i costi per le risorse della sessione.
ALTER SESSION	Eseguire le istruzioni ALTER SESSION.
RESTRICTED SESSION	Collegarsi dopo l'avvio dell'istanza con l'istruzione STARTUP RESTRICT di SQL*Plus.
<b>SYNONYM</b>	
CREATE SYNONYM	Creare sinonimi nello schema del concessionario.
CREATE ANY SYNONYM	Creare sinonimi privati in qualunque schema.
CREATE PUBLIC SYNONYM	Creare sinonimi pubblici.
DROP ANY SYNONYM	Eliminare sinonimi privati in qualsiasi schema.
DROP PUBLIC SYNONYM	Eliminare sinonimi pubblici.
<b>TABLE</b>	
CREATE TABLE	Creare tabelle nello schema del concessionario.
CREATE ANY TABLE	Creare tabelle in qualsiasi schema. Il proprietario dello schema contenente la tabella deve avere quote di spazio sulla tablespace per contenere la tabella.
ALTER ANY TABLE	Modificare qualsiasi tabella o vista in qualunque schema.

(segue)

*(continua)*

<b>PRIVILEGIO</b>	<b>PERMESSO RELATIVO</b>
BACKUP ANY TABLE	Usare l'utility Export per esportare in maniera incrementale gli oggetti dallo schema di altri utenti.
DELETE ANY TABLE	Cancellare righe da tabelle, partizioni di tabelle o viste in qualunque schema.
DROP ANY TABLE	Cancellare o troncare tabelle o partizioni di tabelle in qualsiasi schema.
INSERT ANY TABLE	Inserire righe in tabelle e viste di qualsiasi schema.
LOCK ANY TABLE	Bloccare tabelle e viste in qualsiasi schema.
SELECT ANY TABLE	Eseguire query su tabelle, viste o viste materializzate in qualunque schema.
UPDATE ANY TABLE	Aggiornare righe in tabelle e viste in qualsiasi schema.
<b>TABLESPACE</b>	
CREATE TABLESPACE	Creare tablespace.
ALTER TABLESPACE	Modificare tablespace.
DROP TABLESPACE	Eliminare tablespace.
MANAGE TABLESPACE	Mantenere le tablespace in linea e fuori linea, iniziare e terminare backup di tablespace.
UNLIMITED TABLESPACE	Usare una quantità illimitata di qualunque tablespace. Questo privilegio sostituisce qualsiasi quota specifica assegnata. Se si revoca questo privilegio a un utente, gli oggetti rimarranno nello schema dell'utente ma non sarà possibile allocare altre tablespace a meno che non si sia autorizzati da quote di tablespace specifiche. Non è possibile concedere questo privilegio di sistema ai ruoli.
<b>TRIGGER</b>	
CREATE TRIGGER	Creare un trigger di database nello schema del concessionario.
CREATE ANY TRIGGER	Creare trigger di database in qualsiasi schema.
ALTER ANY TRIGGER	Attivare, disattivare, o compilare trigger di database in qualsiasi schema.
DROP ANY TRIGGER	Eliminare trigger di database in qualunque schema.
ADMINISTER DATABASE TRIGGER	Creare un trigger su DATABASE (si deve disporre anche dei privilegi CREATE TRIGGER o CREATE ANY TRIGGER).
<b>TYPE</b>	
CREATE TYPE	Creare tipi di oggetti e corpi dei tipi di oggetti nello schema del concessionario.
CREATE ANY TYPE	Creare tipi di oggetti e corpi dei tipi di oggetti in qualsiasi schema.
ALTER ANY TYPE	Modificare tipi di oggetti in qualunque schema.
DROP ANY TYPE	Eliminare tipi di oggetti e corpi dei tipi di oggetti in qualsiasi schema.

*(segue)*

(continua)

PRIVILEGIO	PERMESSO RELATIVO
EXECUTE ANY TYPE	Usare e fare riferimento a tipi di oggetti e tipi di collection in qualsiasi schema, e invocare metodi di un tipo di oggetto in qualunque schema se si fa la concessione a un utente specifico. Se si concede EXECUTE ANY TYPE a un ruolo, gli utenti in possesso del ruolo abilitato non saranno in grado di invocare i metodi di un tipo di oggetto in qualsiasi schema.
UNDER ANY TYPE	Creare sottotipi in qualsiasi tipo di oggetto non finale.
<b>USER</b>	
CREATE USER	Creare utenti. Questo privilegio consente al creatore anche di: assegnare quote su qualsiasi tablespace, impostare tablespace predefinite e temporanee, assegnare un profilo come parte di un'istruzione CREATE USER.
ALTER USER	Modificare qualsiasi utente. Questo privilegio autorizza il concessionario a: modificare la password o il metodo di autenticazione di un altro utente, assegnare quote su qualsiasi tablespace, impostare tablespace predefinite e temporanee e assegnare un profilo e ruoli predefiniti.
BECOME USER	Diventare un altro utente. (Richiesto da qualunque utente esegua un'importazione di database completa).
DROP USER	Eliminare utenti.
<b>VIEW</b>	
CREATE VIEW	Creare viste nello schema del concessionario.
CREATE ANY VIEW	Creare viste in qualsiasi schema.
DROP ANY VIEW	Eliminare viste in qualunque schema.
UNDER ANY VIEW	Creare sottoviste in qualsiasi vista oggetto.
<b>VARI</b>	
ANALYZE ANY	Analizzare qualsiasi tabella, cluster o indice in qualsiasi schema.
AUDIT ANY	Porre sotto controllo qualsiasi oggetto in qualunque schema con le istruzioni <i>oggetti_schema</i> di AUDIT.
COMMENT ANY TABLE	Commentare qualunque tabella, vista o colonna in qualsiasi schema.
EXEMPT ACCESS POLICY	Ignorare i controlli di accesso dinamico. Si tratta di un privilegio di sistema molto potente poiché consente al concessionario di ignorare i criteri di sicurezza basati sulle applicazioni. Gli amministratori di database dovrebbero prestare molta attenzione quando concedono questo privilegio.
FORCE ANY TRANSACTION	Indurre il commit o il rollback di qualsiasi transazione distribuita sconosciuta nel database locale. Indurre il fallimento di una transazione distribuita.
FORCE TRANSACTION	Indurre il commit o il rollback delle transazioni distribuite sconosciute del concessionario nel database locale.
GRANT ANY PRIVILEGE	Concedere qualsiasi privilegio di sistema.
RESUMABLE	Abilitare l'allocazione di spazio recuperabile.

(segue)

*(continua)*

PRIVILEGIO	PERMESSO RELATIVO
SELECT ANY DICTIONARY	Eseguire una query su qualsiasi oggetto del dizionario dati nello schema SYS. Questo privilegio consente di sostituire in modo selettivo l'impostazione predefinita FALSE del parametro di inizializzazione O7_DICTIONARY_ACCESSIBILITY. Questo privilegio dev'essere concesso individualmente. Non è incluso in GRANT ALL PRIVILEGES, e non può essere concesso neanche mediante un ruolo.
SYSDBA	Eseguire le operazioni di STARTUP e SHUTDOWN. ALTER DATABASE: aprire, eseguire il MOUNT il backup o modificare il set di caratteri. CREATE DATABASE. ARCHIVELOG e RECOVERY. CREATE SPFILE. Include il privilegio RESTRICTED SESSION.
SYSOPER	Eseguire le operazioni di STARTUP e SHUTDOWN. ALTER DATABASE OPEN   MOUNT   BACKUP ARCHIVELOG e RECOVERY. CREATE SPFILE. Include il privilegio RESTRICTED SESSION.

**NOTA** Per le tabelle esterne, gli unici privilegi validi sono CREATE ANY TABLE, ALTER ANY TABLE, DROP ANY TABLE e SELECT ANY TABLE.

I privilegi di oggetto possono essere applicati solo ad alcuni tipi di oggetti. La tabella seguente mostra le relazioni e i privilegi di oggetti che si possono concedere.

PRIVILEGIO OGGETTO	TABELLA	VISTA	SEQUENZA	PROCEDURE, FUNZIONI, PACKAGE <sup>a</sup>	VISTA MATERIALIZZATA	DIRECTORY	LIBRERIA	TIPO DEFINITO DALL'UTENTE	OPERATORE	TIPO DI INDICE
ALTER	X		X							
DELETE	X	X			X <sup>b</sup>					
EXECUTE				X			X	X	X	X
INDEX	X									
INSERT	X	X			X <sup>b</sup>					
ON COMMIT REFRESH	X									
QUERY REWRITE	X									
READ						X				
REFERENCES	X	X								
SELECT	X	X	X		X					
UNDER		X						X		
UPDATE	X	X			X <sup>b</sup>					
WRITE						X				

<sup>a</sup> Oracle tratta una classe, un'origine o una risorsa Java come se fosse una procedura per concedere privilegi sugli oggetti.

<sup>b</sup> I privilegi DELETE, INSERT e UPDATE possono essere concessi solo a viste materializzate aggiornabili.

## PRO\*C

Il PRO\*C è un'estensione del C che permette di sviluppare user exit e altri programmi di accesso al database Oracle. Un precompilatore converte il codice PRO\*C in codice C normale, pronto per essere compilato. Vedere *Pro\*C/C++ Precompiler Programmers's Guide*.

## PROCEDURA

Una procedura è un insieme di istruzioni (solitamente combinazioni di comandi SQL e PL/SQL) salvate per esecuzioni ripetute e esecuzioni di chiamata. Vedere CREATE PROCEDURE.

## PROCESSO PMON

Il Process MONitor è un processo in background utilizzato per recuperare le situazioni in cui un processo si interrompe durante l'accesso al database. Vedere BACKGROUND, PROCESSO.

## PRODUCT\_USER\_PROFILE

PRODUCT\_USER\_PROFILE è una tabella SYSTEM di Oracle utilizzata per limitare l'uso di singoli prodotti Oracle o per disattivare uno o più comandi disponibili nel prodotto. Vedere *SQL\*Plus User's Guide and Reference*.

## PROFILE

Un profilo è una collezione di impostazioni Oracle che limitano le risorse del database. Vedere CREATE PROFILE e il Capitolo 19.

## PROMPT

**VEDERE ANCHE** ACCEPT.

### SINTASSI

PROMPT [*testo*]

**DESCRIZIONE** Visualizza il *testo* sullo schermo dell'utente. Se non si specifica alcun *testo*, viene visualizzata una linea vuota. Per visualizzare una variabile, vedere PRINT.

## PSEUDOCOLONNE

Una pseudocolonna è una “colonna” che produce un valore quando selezionata ma non è una vera colonna della tabella. Esempi tipici sono RowID e SysDate. Ecco le pseudocolonne supportate attualmente da Oracle.

PSEUDOCOLONNA	VALORE RESTITUITO
sequence.CurrVal	Il valore corrente della sequenza specificata.
Level	Pari a 1 per un nodo principale, a 2 per un figlio di un nodo e così via. Indica quanti livelli dell'albero sono stati analizzati.
sequence.NextVal	Il valore successivo della sequenza indicata. Incrementa la sequenza automaticamente.
NULL	Un valore NULL.
RowID	Restituisce il numero dell'identificativo di una riga. RowID viene utilizzato in UPDATE... WHERE e SELECT... FOR UPDATE. In questo modo, si garantisce che vengano aggiornate solo le righe indicate.
RowNum	Restituisce il numero di sequenza in cui è stata restituita una riga quando selezionata da una tabella. Il RowNum della prima riga è 1, quello della seconda è 2 e così via.

## PUBLIC

Public può avere queste due definizioni:

- Qualcosa che sia stato reso pubblico è visibile o disponibile per tutti gli utenti. Possono essere pubblici i sinonimi e i database link. In Oracle, per rendere pubblico qualcosa un utente deve possedere i privilegi CREATE PUBLIC SYNONYM o CREATE PUBLIC DATABASE LINK. Gli utenti possono inoltre concedere l'accesso ai propri oggetti agli altri utenti.
- Un gruppo cui appartiene qualsiasi utente di un dato database, il nome di tale gruppo.

## PUBLIC SYNONYM

Un sinonimo pubblico è un sinonimo di un oggetto di database che un utente ha creato, tramite il privilegio CREATE PUBLIC SYNONYM, per essere utilizzato da tutti gli utenti di Oracle.

## QUERY

Una query è un'istruzione SQL usata per recuperare dati da una o più tabelle (o viste). Le query iniziano con la parola chiave **select**.

## QUERY AD ALBERO

Una query ad albero è una query il cui risultato evidenzia relazioni gerarchiche tra le righe di una tabella. *Vedere* CONNECT BY.

## QUERY CORRELATA

Una query correlata è una subquery che viene eseguita ripetutamente, una volta per ciascun valore di una riga candidata selezionata dalla query principale. L'esito di ogni esecuzione della subquery dipende dai valori di uno o più campi all'interno della riga candidata; in altre parole, la subquery è correlata alla query principale. *Vedere* il Capitolo 12.

## QUERY DISTRIBUITA

Una query distribuita è una query che seleziona dati da più database, solitamente residenti su più nodi di una rete.

## QUERY FLASHBACK

Una query flashback mostra i dati così come erano prima di un commit. Non è possibile eseguire DML quando ci si trova nella modalità di query flashback. Per esempi e restrizioni applicate alle query flashback si rimanda al Capitolo 26.

## QUERY PADRE

La query padre è la query più esterna (quella che visualizza il risultato) in una query principale che contiene una subquery. *Vedere QUERY PRINCIPALE.*

## QUERY PRINCIPALE

Una query principale è la query più esterna o iniziale di una query contenente una subquery. È la query le cui colonne producono il risultato finale.

## QUIT

**VEDERE ANCHE** COMMIT, DISCONNECT, EXIT, SET AUTOCOMMIT, START.

### SINTASSI

QUIT

**DESCRIZIONE** QUIT termina una sessione SQL\*PLUS e riporta l'utente al sistema operativo, al programma chiamante o al menu precedente.

## QUOTA

Una quota è un limite imposto a una risorsa. Le quote possono limitare la quantità di memoria utilizzata da ciascun utente del database. *Vedere CREATE USER e ALTER USER.*

## RADICE

In una tabella ad albero, la radice è l'origine dell'albero. Si tratta di una riga che non ha padre e i cui figli, nipoti ecc. costituiscono l'intero albero. Nelle query ad albero, la radice è la riga specificata nella clausola START WITH.

## RAISE

**VEDERE ANCHE** DECLARE EXCEPTION, EXCEPTION, EXCEPTION\_INIT.

## SINTASSI

RAISE [exception]

**DESCRIZIONE** RAISE assegna un nome al flag dell’eccezione che si intende sollevare, in base a una condizione verificata. L’eccezione deve essere stata dichiarata esplicitamente o appartene-re alle eccezioni di sistema come DIVIDE\_BY\_ZERO (per un elenco completo, vedere EXCEPTION).

L’istruzione RAISE sposta il controllo alla sezione EXCEPTION del blocco corrente nella quale si dovrà eseguire un controllo per verificare quale eccezione è stata sollevata. Se non è presente alcuna sezione EXCEPTION, il controllo salta alla sezione EXCEPTION più vicina di un blocco di inclusione (praticamente, ripercorrendo al contrario l’annidamento dei blocchi). Se non viene trovata alcuna logica per la gestione delle eccezioni, il controllo passa da PL/SQL al programma o all’ambiente chiamante fornendo un errore di eccezione non gestibile. Questo può essere evitato utilizzando OTHERS. Vedere EXCEPTION.

RAISE può essere utilizzato senza parametri in un solo caso: all’interno di una sezione EXCEPTION per fare in modo che l’eccezione corrente venga gestita dalla sezione EXCEPTION del blocco più esterno, invece che dal blocco corrente. Se per esempio si fosse verificato un errore NOT\_LOGGED\_ON e la sezione EXCEPTION locale contenesse le righe seguenti:

```
when not_logged_on
  then raise;
```

il controllo passerebbe alla sezione EXCEPTION del blocco immediatamente più esterno, oppure al programma se non esistesse un blocco simile. Questo blocco EXCEPTION potrebbe essere verificato per controllare la presenza di eventuali errori NOT\_LOGGED\_ON. Il vantaggio con-siste nel fatto che per una certa classe di errori, soprattutto quelli in cui le fasi di recupero, traccia e registrazione degli errori che si desiderano eseguire possono essere ampie, è possibile impostare la sezione EXCEPTION di ogni blocco annidato per demandare semplicemente l’eccezione a un unico blocco EXCEPTION.

## RAMI

I rami rappresentano una serie di nodi interconnessi di un albero logico. Vedere CONNECT BY.

## RATIO\_TO\_REPORT

**VEDERE ANCHE** FUNZIONI DI GRUPPO.

### SINTASSI

RATIO\_TO\_REPORT ( *expr* ) OVER ( [clausola\_partizionamento\_query] )

**DESCRIZIONE** RATIO\_TO\_REPORT è una funzione analitica. Calcola il rapporto di un valore rispetto alla somma di un gruppo di valori. Se *expr* vale NULL, anche il valore di questa funzione sarà NULL.

## RAW, TIPO DI DATI

Una colonna RAW contiene dati binari in un qualsiasi formato compatibile con il computer host. Le colonne RAW sono utili per memorizzare dati binari (non corrispondenti a caratteri).

## RAWTOHEX

**VEDERE ANCHE** HEXTORAW.

### SINTASSI

RAWTOHEX(*stringa\_binaria*)

**DESCRIZIONE** RAWTOHEX converte una stringa di numeri binari in una stringa di caratteri di cifre esadecimali.

## RAWTONHEX

**VEDERE ANCHE** RAWTOHEX.

### SINTASSI

RAWTONHEX( *raw* )

**DESCRIZIONE** RAWTONHEX converte un dato RAW in un valore NVARCHAR2 contenente l'equivalente esadecimale.

## RDBMS

*Vedere* RELAZIONALE, SISTEMA DI GESTIONE DI DATABASE.

## RECORD

Record è un sinonimo di riga.

## RECORD (PL/SQL)

**VEDERE ANCHE** TABELLA (PL/SQL); DATI, TIPI.

### SINTASSI

```
TYPE nuovo_tipo IS RECORD  
  (campo {tipo | tabella.colonna%TYPE} [NOT NULL]  
  [, campo {tipo | tabella.colonna%TYPE} [NOT NULL] ...]);
```

**DESCRIZIONE** La dichiarazione di un RECORD produce un nuovo tipo di dati che può essere utilizzato per dichiarare le nuove variabili. I singoli componenti di un record si dicono campi e ciascuno di essi è di un tipo di dati opportuno. Questi tipi di dati possono essere tipi standard di PL/SQL (compresi altri RECORD ma escludendo le tabelle), o un riferimento al tipo di una particolare colonna di una data tabella. Ogni campo può possedere anche l'attributo NOT NULL secondo cui il campo deve sempre avere un valore significativo.

Si può fare riferimento ai singoli campi di un record con la classica notazione separata da punti.

## RECOVER

**VEDERE ANCHE** ALTER DATABASE, Capitolo 40.

## SINTASSI

RECOVER {*generale* | *gestito* | END BACKUP}

dove la clausola *generale* ha la sintassi seguente:

```
[AUTOMATIC] [FROM locazione]
{ { recupero_completo_database | recupero_parziale_database |LOGFILE filename}
[ {TEST | ALLOW integer CORRUPTION } [TEST | ALLOW intero CORRUPTION ]...]
|CONTINUE [DEFAULT]|CANCEL}
```

dove la clausola *recupero\_completo\_database* presenta la sintassi seguente:

```
[STANDBY] DATABASE
[ {UNTIL {CANCEL | TIME data | CHANGE intero} | USING BACKUP CONTROLFILE}
[UNTIL {CANCEL | TIME data | CHANGE intero} | USING BACKUP CONTROLFILE]...]
```

dove la clausola *recupero\_parziale\_database* ha la sintassi seguente:

```
{TABLESPACE tablespace [. tablespace] ... | DATAFILE nome_file_dati [. nome_file_dati]...
| STANDBY {TABLESPACE tablespace [. tablespace] ... | DATAFILE nome_file_dati [. nome_file_dati]...}
UNTIL [CONSISTENT] [WITH] CONTROLFILE }
```

mentre la clausola *gestito* presenta questa sintassi:

```
MANAGED STANDBY DATABASE
[ {Nodelay | [TIMEOUT] intero | CANCEL [IMMEDIATE] [NOWAIT]}
| [DISCONNECT [FROM SESSION] ] [FINISH [NOWAIT] ] ]
```

**DESCRIZIONE** Il comando RECOVER, come la clausola RECOVER del comando ALTER DATABASE recupera un database utilizzando diverse opzioni. Il recupero AUTOMATIC genera automaticamente i nomi dei redo log file durante il recupero stesso. La clausola FROM specifica la locazione del gruppo di redo log file archiviati e deve essere un nome di file completamente qualificato. Il valore predefinito viene impostato dal parametro di inizializzazione LOG\_ARCHIVE\_DEST.

L'opzione DATABASE ripristina completamente il database ed è l'opzione predefinita. L'alternativa UNTIL CANCEL recupera il database finché non si annulla il recupero con l'opzione CANCEL. L'alternativa UNTIL TIME esegue l'operazione di recupero fino a una certa data e ora. L'alternativa UNTIL CHANGE recupera fino a un *numero di cambiamenti di sistema* (un numero unico assegnato a ciascuna transazione) specificato da un numero intero. L'alternativa USING BACKUP CONTROLFILE esegue il recupero applicando il redo log a un control file del backup.

## RECOVERY MANAGER

Recovery Manager (RMAN) è un prodotto della Oracle Corporation. RMAN è utilizzato dai DBA per eseguire i backup del database. Le informazioni sui backup sono memorizzate in un database repository di RMAN o nei control file dei database per cui si esegue il backup.

## RECSEP (SQL\*PLUS)

Vedere SET.

## RECSEPCHAR (SQL\*PLUS)

Vedere SET.

## RECUPERO DI DISCHI

Si tratta dell'operazione di recupero di guasti hardware che possono pregiudicare lettura, scrittura dei dati e operazioni simili nel database. *Vedere anche* RECUPERO DI ISTANZA.

## RECUPERO DI ISTANZA

Il recupero di un'istanza è il processo di recupero del guasto (hardware o software) subito dall'istanza stessa. Si ha quando il software ha terminato la propria esecuzione in maniera drastica a causa di un errore, di un guasto o di un'interazione distruttiva tra programmi. *Vedere anche* RECUPERO DI DISCHI.

## REDO LOG

Un redo log è un log sequenziale di azioni eseguite nel database. Il log è costituito sempre almeno da due file; uno può essere facoltativamente archiviato mentre l'altro viene scritto. Quando il file in cui si scrive è pieno, si comincia a riutilizzare l'altro.

## REDO LOG IN LINEA

I redo log in linea sono redo log file non ancora archiviati. Possono essere aperti per tenere traccia di un'attività di registrazione, oppure sono stati scritti in precedenza e si trovano semplicemente in attesa di archiviazione.

## REDO LOG SEQUENCE NUMBER

Il redo log sequence number è un numero utilizzato per identificare un redo log, impiegato quando si applica il file di log alle operazioni di recupero.

## REF

Un tipo di dati di riferimento utilizzato con tabelle oggetto. *Vedere* il Capitolo 33.

## REFTOHEX

**VEDERE ANCHE** Capitolo 33.

### SINTASSI

REFTOHEX ( *expr* )

**DESCRIZIONE** REFTOHEX converte l'argomento *expr* in un valore contenente il suo equivalente esadecimale. *expr* deve restituire un REF.

## **REGOLA DI INTEGRITÀ REFERENZIALE**

Una regola di integrità referenziale è un vincolo di integrità che impone l'integrità referenziale.

## **RELAZIONALE A OGGETTI, SISTEMA DI GESTIONE DI DATABASE**

Un sistema di gestione di database relazionale a oggetti (ORDBMS) supporta sia le caratteristiche dei database relazionali puri (come le chiavi primarie ed esterne) sia quelle dei database a oggetti (come ereditarietà e encapsulamento). Per una descrizione dell'implementazione di un ORDBMS di Oracle, si consulti il Capitolo 4.

## **RELAZIONALE, SISTEMA DI GESTIONE DI DATABASE**

Un RDBMS è un programma improntato alla memorizzazione e al recupero di dati di qualsiasi tipo, e che organizza i dati in tabelle composte da una o più unità di informazione (righe), ciascuna contenente lo stesso insieme di tipi di dati (colonne). Oracle è un sistema di gestione di database relazionale.

## **RELAZIONE**

*Vedere TABELLA* e il Capitolo 1.

## **REMARK**

**VEDERE ANCHE** /\* \*/, --, Capitolo 6.

### **SINTASSI**

REM[ARK] *testo*

**DESCRIZIONE** REMARK inizia una singola linea di commenti, solitamente di documentazione, in un file di avvio. Non viene interpretato come un comando. REMARK non può comparire all'interno di un'istruzione SQL.

### **ESEMPIO**

REM Riduce la larghezza predefinita delle colonne per questo campo di dati:  
column ActionDate format a6 trunc

## **RENAME**

**VEDERE ANCHE** COPY, CREATE SEQUENCE, CREATE SYNONYM, CREATE TABLE, CREATE VIEW.

### **SINTASSI**

RENAME *vecchio* TO *nuovo*;

**DESCRIZIONE** RENAME cambia il nome di una tabella, vista, sequenza o sinonimo assegnando loro un nome nuovo. Non sono necessari apici attorno ai nomi. Il nuovo nome non dev'essere né una parola riservata né il nome di un oggetto già esistente di proprietà dell'utente stesso.

## ESEMPIO

La riga seguente cambia il nome della tabella BIBLIOTECA in LIBRO:

```
rename BIBLIOTECA to LIBRO;
```

## REPFOOTER

**VEDERE ANCHE** ACCEPT, BTITLE, DEFINE, PARAMETRI, REPHEADER, Capitolo 14.  
**SINTASSI**

```
REPF[OOTER] [PAGE] [opzione [testo|variabile]... | OFF|ON]
```

**DESCRIZIONE** REPFOOTER (REPort FOOTER) inserisce un pi  di pagina (magari a pi  linee) nell'ultima pagina di un rapporto. OFF e ON rispettivamente sopprimono e ripristinano la visualizzazione del testo senza modificarne i contenuti. REPFOOTER da solo mostra le opzioni correnti e il valore di *testo* o *variabile*. PAGE inizia una nuova pagina prima di visualizzare il pi  di pagina del report specificato.

*testo*   il pi  di pagina che si desidera assegnare a questo report, e *variabile*   il nome di una variabile definita dall'utente o gestita dal sistema come SQL.LNO, il numero di linea corrente; SQL.PNO, il numero di pagina corrente; SQL.RELEASE, il numero della release corrente di Oracle; SQL.SQLCODE, il codice di errore corrente e SQL.USER, il nome utente.

Le opzioni possibili sono elencate di seguito.

- COL *n* salta direttamente alla posizione *n* (dal margine sinistro) della linea corrente.
- S[KIP] *n* visualizza *n* linee vuote. Se non viene specificato alcun numero, viene stampata una sola linea vuota. Se *n* vale 0, non viene visualizzata alcuna linea vuota e la posizione di stampa corrente diventa la prima della linea corrente (all'estremit  sinistra della pagina).
- TAB *n* salta di *n* tabulazioni verso destra (verso sinistra se *n*   negativo).
- LE[FT], CE[NTER] e R[IGHT] giustificano a sinistra, centrano e giustificano a destra i dati della linea corrente. Eventuali stringhe di testo e variabili che seguissero questi comandi verrebbero giustificate anch'esse, fino alla fine del comando. CENTER e RIGHT utilizzano il valore impostato da SET LINESIZE per determinare dove disporre il testo e le variabili.
- BOLD visualizza i dati in grassetto. SQL\*PLUS rappresenta il grassetto sul terminale ripetendo i dati su tre linee consecutive.
- FORMAT stringa specifica il modello del formato per la visualizzazione del testo o delle variabili seguenti e segue la stessa sintassi dell'opzione FORMAT del comando COLUMN; per esempio, FORMAT A12 o FORMAT \$999,990.99. Ogni volta che compare FORMAT, esso scavalca il precedente formato attivo fino a quel momento. Se non si specifica alcun modello per FORMAT, viene utilizzato quello impostato da SET NUMFORMAT. Se NUMFORMAT non   stato definito, viene utilizzato il modello predefinito per SQL\*PLUS.

Le date vengono visualizzate in base al formato predefinito a meno che la data di una variabile non sia stata convertita mediante la funzione TO\_CHAR.

## REPHEADER

**VEDERE ANCHE** ACCEPT, BTITLE, DEFINE, PARAMETRI, REPFOOTER, Capitolo 14.

## SINTASSI

REPH[EADER] [PAGE] [*opzione* [*testo|variabile*]... | OFF|ON]

**DESCRIZIONE** REPHEADER (REPort HEADER) inserisce un'intestazione (magari a più linee) nella prima pagina di un report. OFF e ON rispettivamente eliminano e ripristinano la visualizzazione del testo senza modificarne i contenuti. REPHEADER da solo mostra le opzioni correnti e il valore di *testo* o *variabile*. PAGE inizia una nuova pagina dopo aver visualizzato l'intestazione del report.

*testo* è un'intestazione che si desidera assegnare a questo report, mentre *variabile* è il nome di una variabile definita dall'utente o gestita dal sistema come SQL.LNO, il numero di linea corrente; SQL.PNO, il numero di pagina corrente; SQL.RELEASE, il numero della release corrente di Oracle; SQL.SQLCODE, il codice di errore corrente e SQL.USER, il nome utente.

Le opzioni possibili sono elencate di seguito.

- COL *n* salta direttamente alla posizione *n* (dal margine sinistro) della linea corrente.
- S[KIP] *n* visualizza *n* linee vuote. Se non viene specificato alcun numero, viene stampata una sola linea vuota. Se *n* vale 0, non viene visualizzata alcuna linea vuota e la posizione di stampa corrente diventa la prima della linea corrente (all'estremità sinistra della pagina).
- TAB *n* salta di *n* tabulazioni verso destra (verso sinistra se *n* è negativo).
- LE[FT], CE[NTER] e R[IGHT] giustificano a sinistra, centrano e giustificano a destra i dati della linea corrente. Eventuali stringhe di testo e variabili che seguissero questi comandi verrebbero giustificate anch'esse, fino alla fine del comando. LEFT, CENTER, RIGHT o COL.CENTER e RIGHT utilizzano il valore impostato da SET LINESIZE per determinare dove disporre il testo e le variabili.
- BOLD visualizza i dati in grassetto. SQL\*PLUS rappresenta il grassetto sul terminale ripetendo i dati su tre linee consecutive.
- FORMAT stringa specifica il modello del formato per la visualizzazione del testo o delle variabili seguenti e segue la stessa sintassi dell'opzione FORMAT del comando COLUMN; per esempio, FORMAT A12 o FORMAT \$999,990.99. Ogni volta che compare FORMAT, esso scavalca il precedente formato attivo fino a quel momento. Se non si specifica alcun modello per FORMAT, viene utilizzato quello impostato da SET NUMFORMAT. Se NUMFORMAT non è stato definito, viene utilizzato il modello predefinito per SQL\*PLUS.

Le date vengono visualizzate in base al formato predefinito a meno che la data di una variabile non sia stata convertita mediante la funzione TO\_CHAR.

## REPLACE

**VEDERE ANCHE** CARATTERI, FUNZIONI, TRANSLATE, Capitolo 16.

### SINTASSI

REPLACE(*stringa,if,then*)

**DESCRIZIONE** REPLACE sostituisce uno o più caratteri di una stringa con 0 o più caratteri. *if* è una stringa di uno o più caratteri. Ogni volta che compare all'interno di *stringa*, viene sostituito dal contenuto di *then*.

### ESEMPIO

```
REPLACE('GEORGE','GE',null) = OR
REPLACE('BOB','BO','TA') = TAB
```

## REPLICA

La replica è quel processo che prevede la copia dei dati da un database in un altro. La destinazione in cui vengono copiati i dati replicati prende il nome di replica. Per gestire il processo di replica, si dovrà stabilire:

- la collocazione, la struttura e la proprietà dei dati di origine;
- la frequenza degli aggiornamenti per i dati di origine;
- i requisiti di opportunità per i dati delle replica;
- la volatilità dei dati di origine;
- i vari impieghi della replica.

Se la replica è anche in grado di apportare delle modifiche e queste ultime devono essere riportate al database originale, allora si avrà una configurazione *con più siti principali* e si dovrà gestire la risoluzione dei conflitti e gli errori per questo ambiente. In generale, la proprietà a singolo sito dei dati con più repliche di sola lettura è più semplice da gestire di una configurazione con più siti principali.

Vedere CREATE MATERIALIZED VIEW, CREATE DATABASE LINK

## RESOURCE (RISORSA)

Risorsa è il termine generale che denota un oggetto logico o una struttura fisica di un database. Le risorse possono essere bloccate. Quelle che gli utenti possono bloccare direttamente sono le tabelle e le righe. Le risorse bloccabili dall'RDBMS sono molteplici e comprendono tabelle dei dizionari dati, cache e file.

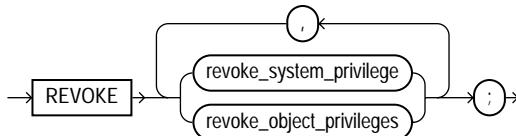
RESOURCE è anche il nome di un ruolo standard di Oracle. Vedere RUOLO.

## REVOKE

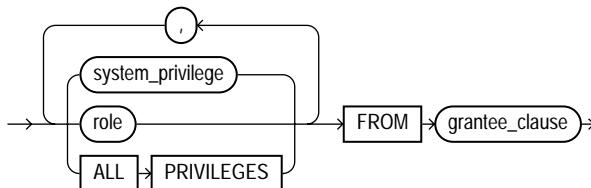
**VEDERE ANCHE** DISABLE, ENABLE, GRANT, PRIVILEGI, Capitolo 19.

### SINTASSI

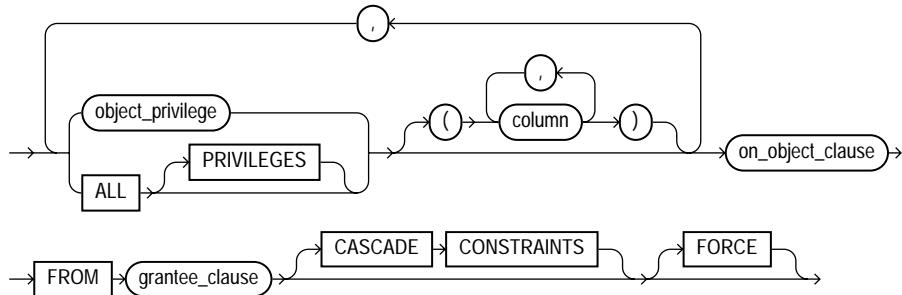
revoke



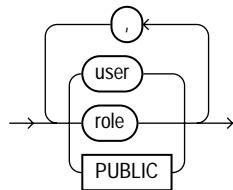
revoke\_system\_privileges



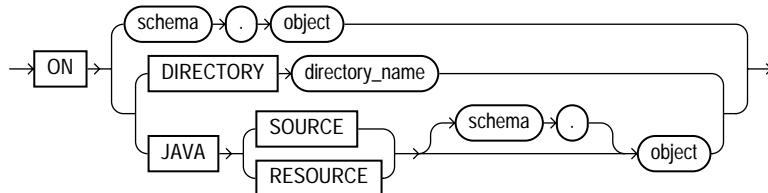
## revoke\_object\_privileges



## grantee\_clause



## on\_object\_clause



**DESCRIZIONE** Il comando REVOKE rimuove privilegi e ruoli da utenti oppure privilegi da ruoli. Qualsiasi privilegio di sistema può essere rimosso; vedere PRIVILEGI.

Quando si revoca un privilegio a un utente, questi non potrà più eseguire le operazioni ammesse da quel privilegio. Se si revoca un privilegio a un ruolo, nessun utente con quel ruolo è più in grado di eseguire le operazioni relative al privilegio, a meno che non gli venga concesso il permesso di farlo da un altro ruolo o direttamente in quanto utente. Se si revoca un privilegio a PUBLIC, nessun utente cui era stato concesso quel privilegio tramite PUBLIC potrà eseguire le operazioni consentite dal privilegio.

Se si revoca un ruolo a un utente, questo non potrà più abilitarlo (vedere ENABLE) ma potrà continuare a esercitare il privilegio nella sessione corrente. Se si revoca un ruolo a un altro ruolo, gli utenti cui era stato concesso questo ruolo non potranno più abilitarlo, ma potranno continuare a usare il privilegio durante la sessione corrente. Se si revoca un ruolo a PUBLIC, gli utenti cui è stato concesso il ruolo tramite PUBLIC non potranno più abilitarlo.

Per quanto riguarda l'accesso agli oggetti, il comando REVOKE rimuove i privilegi relativi a un oggetto da un utente o da un ruolo. Se i privilegi di un utente vengono revocati, l'utente non può eseguire le operazioni consentite dal privilegio su quel determinato oggetto. Se vengono revocati i privilegi di un ruolo, nessun utente con quel ruolo potrà eseguire le operazioni relative

a quei privilegi, a meno che questi non vengano concessi tramite un altro ruolo oppure direttamente all'utente. Se si revocano i privilegi di PUBLIC, nessun utente cui è stato concesso il privilegio tramite PUBLIC potrà eseguire le operazioni ad esso associate.

La clausola CASCADE CONSTRAINTS elimina qualsiasi vincolo di integrità referenziale definito da uno o più utenti in possesso del ruolo. Questo vale per il privilegio REFERENCES.

## RIGA

Riga può avere due definizioni:

- Un insieme di campi di una tabella; per esempio i campi che rappresentano un impiegato nella tabella LAVORATORE.
- Un insieme di campi prodotti da una query. In questo caso RECORD è un sinonimo.

## ROLL FORWARD

Si tratta di un meccanismo per applicare nuovamente le modifiche apportate al database. Talvolta serve per il recupero dei dischi e talvolta per il recupero dell'istanza. Il redo log contiene le voci redo utilizzate per il roll forward.

## ROLLBACK

Il rollback annulla una parte o l'intero lavoro effettuato nel corso della transazione corrente (fino all'ultimo COMMIT o SAVEPOINT).

### ROLLBACK (Forma 1, SQL)

**VEDERE ANCHE** COMMIT, SET AUTOCOMMIT, Capitolo 15.

#### SINTASSI

```
ROLLBACK [WORK] [ TO [SAVEPOINT] savepoint | FORCE "testo"];
```

**DESCRIZIONE** ROLLBACK annulla tutti i cambiamenti apportati alle tabelle del database dall'ultimo salvataggio e toglie qualsiasi blocco applicato alle tabelle. Ogni volta che si interrompe una transazione, a causa di un errore di esecuzione o di un problema all'alimentazione viene generato un rollback automatico. ROLLBACK non influisce solo sull'ultima istruzione insert, update o delete, ma su tutte quelle occorse dall'ultimo COMMIT. Questo permette di trattare i vari spezzoni di lavoro come una singola unità e salvare solo quando tutte le modifiche sono state effettuate.

Se si specifica un SAVEPOINT, ROLLBACK salta a un punto specifico (etichettato) della sequenza di transazioni elaborate, il SAVEPOINT. Cancella qualsiasi SAVEPOINT temporaneo e annulla ogni blocco di tabella o riga occorso dopo la creazione del SAVEPOINT.

Se si specifica FORCE, ROLLBACK effettua un rollback automatico di una transazione sconosciuta identificata dal testo che corrisponde a un ID di transazione locale o globale dalla vista del dizionario dati DBA\_2PC\_PENDING.

**NOTA** Se SET AUTOCOMMIT è attivo, ogni inserimento, aggiornamento o cancellazione salva immediatamente e automaticamente le eventuali modifiche nel database. L'inserimento della parola ROLLBACK porta alla visualizzazione di un messaggio 'ROLLBACK COMPLETE' che però non significa nulla poiché ROLLBACK esegue il rollback solo fino all'ultimo COMMIT.

**NOTA** Tutti i comandi DDL provocano un COMMIT.

## ROLLBACK (Forma 2, Embedded SQL)

**VEDERE ANCHE** COMMIT, SAVEPOINT, SET TRANSACTION, guida di programmazione del precompilatore.

### SINTASSI

```
EXEC SQL [AT {database | :variabile }]  
ROLLBACK [WORK]  
[TO [SAVEPOINT] savepoint] | [FORCE testo] | [RELEASE] ]
```

**DESCRIZIONE** ROLLBACK termina la transazione corrente, annulla qualunque modifica risultante dalla transazione in corso e rilascia ogni blocco, ma non modifica né le variabili host né il flusso di esecuzione del programma. *database* indica il nome del database su cui ROLLBACK dovrebbe operare. Nel caso non venga indicato, si utilizza il database predefinito dell'utente. SAVEPOINT consente di eseguire un rollback su un determinato SAVEPOINT che sia stato dichiarato precedentemente (*vedere* SAVEPOINT).

FORCE effettua il rollback manuale di una transazione sconosciuta, specificata dal testo contenente l'ID di transazione dalla vista del dizionario dati DBA\_2PC\_PENDING. WORK è assolutamente facoltativo e viene utilizzato solo per migliorare la leggibilità.

ROLLBACK e COMMIT dispongono entrambi delle opzioni RELEASE. RELEASE dev'essere specificata da ROLLBACK o da COMMIT dopo l'ultima transazione, altrimenti i blocchi applicati durante il programma possono interferire col lavoro degli altri utenti. Si ha un ROLLBACK automatico (comprensivo di RELEASE) qualora il programma non si concluda normalmente.

## ROLLUP

**VEDERE ANCHE** GROUPING, CUBE, Capitolo 13.

### SINTASSI

```
GROUP BY ROLLUP(colonna1, colonna2)
```

**DESCRIZIONE** ROLLUP genera totali parziali per gruppi di righe all'interno di una query. Vedere il Capitolo 13.

## ROUND (Forma 1, per date)

**VEDERE ANCHE** DATE, FUNZIONI; ROUND (NUMERI); TRUNC; Capitolo 9.

### SINTASSI

```
ROUND (data, 'formato')
```

**DESCRIZIONE** ROUND arrotonda una data in base al formato specificato. Senza un formato, la data viene arrotondata al giorno successivo se l'orario coincide con 12:00:00 P.M. (mezzogiorno esatto) o più tardi, oppure alla data odierna se è prima di mezzogiorno; in altre parole, viene arrotondata al giorno più vicino. L'ora della data così ottenuta viene impostata a 12 A.M., l'inizio di una giornata.

### Formati disponibili per gli arrotondamenti

FORMATO	SIGNIFICATO
CC,SCC	Secolo (arrotonda al primo gennaio del secolo successivo, a partire dalla mezzanotte esatta del primo gennaio del 1950, 2050 e così via).
SYEAR,SYYY,Y,YY,YYYY,YYYY	Anno (arrotonda al primo gennaio dell'anno successivo, a partire dalla mezzanotte esatta del primo luglio).
IYYY, IYY, IY, I	Anno ISO.
Q	Trimestre (arrotonda a partire dalla mezzanotte esatta del 16 del secondo mese del trimestre, indipendentemente dal numero di giorni del mese).
MONTH,MON,MM,RM	Mese (arrotonda a partire dalla mezzanotte esatta del 16 del mese, a prescindere dal numero di giorni del mese).
WW	Giorno della settimana che coincide con il primo giorno dell'anno.
IW	Giorno della settimana che coincide con il primo giorno dell'anno ISO.
W	Arrotonda al giorno più vicino a quello che coincide con il primo giorno del mese.
DDD,DD,J	Arrotonda a mezzogiorno esatto del giorno successivo. È il formato predefinito se non ne viene specificato un altro.
DAY,DY,D	Arrotonda alla domenica successiva (primo giorno della settimana) a partire da mezzogiorno esatto di mercoledì.
HH,HH12,HH24	Arrotonda all'ora intera successiva, a partire da 30 minuti e 30 secondi dopo l'ora.
MI	Arrotonda al minuto intero successivo, a partire da 30 secondi di questo minuto.

Quando ww e w eseguono un arrotondamento, l'ora della data arrotondata viene confrontata con una data impostata a 12 A.M., l'inizio di una giornata. Il risultato di ROUND è una data nuova, impostata anch'essa a mezzanotte.

### ROUND (Form 2, per numeri)

**VEDERE ANCHE** CEIL; FLOOR; NUMERI, FUNZIONI; ROUND (DATE); TRUNC; Capitolo 8.

### SINTASSI

ROUND(*valore, precisione*)

**DESCRIZIONE** ROUND arrotonda *valore a precisione*. *precisione* è un intero positivo, negativo o uguale a zero. Un numero intero negativo arrotonda il valore al numero di cifre indicato a sinistra del punto decimale. Un numero intero positivo arrotonda il valore al numero di cifre indicato a destra del punto decimale.

## ESEMPIO

ROUND(123.456,0) = 123  
 ROUND(123.456,-2) = 100  
 ROUND(-123.456,2) = -123.46

## ROW\_NUMBER

**VEDERE ANCHE** ROWNUM.

### SINTASSI

ROW\_NUMBER ( ) OVER ( [clausola\_partizionamento\_query] order\_by\_clausola )

**DESCRIZIONE** ROW\_NUMBER è una funzione analitica. Assegna un numero unico a ciascuna riga cui viene applicata (a ogni riga della partizione oppure a ogni riga restituita dalla query), nella sequenza ordinata di righe specificata in *order\_by\_clausola*, che inizia con 1.

## ROWID

ROWID è l'indirizzo logico di una riga, pertanto è unico all'interno del database.

ROWID può essere selezionato o impiegato in una clausola where ma non può essere modificato da inserimenti, aggiornamenti o cancellazioni. Non si tratta di una vera colonna di dati; è semplicemente un indirizzo logico creato sulla base di informazioni interne al database. ROWID è utile all'interno di clausole where per aggiornare o cancellare rapidamente delle righe. ROWID può variare se la tabella che lo contiene viene esportata e importata.

Il tipo di dati ROWID è una stringa esadecimale che rappresenta l'indirizzo univoco di una riga di una tabella. Questo tipo di dati è utilizzato soprattutto per i valori restituiti dalla pseudocolonna ROWID.

## ROWIDTOCHAR

**VEDERE ANCHE** CHARTOROWID, CONVERSIONE, FUNZIONI.

### SINTASSI

ROWIDTOCHAR(*RowId*)

**DESCRIZIONE** ROWIDTOCHAR trasforma un RowID interno di Oracle in modo che si comporti come una stringa di caratteri. Dato però che Oracle esegue questo tipo di conversione automaticamente, questa funzione non è realmente necessaria. Sembra piuttosto uno strumento di debug che ha trovato il modo di rendersi disponibile a tutti gli utenti.

## ROWIDTONCHAR

**VEDERE ANCHE** CHARTOROWID, ROWIDTOCHAR, CONVERSIONE, FUNZIONI.

### SINTASSI

ROWIDTONCHAR(*RowId*)

**DESCRIZIONE** ROWIDTONCHAR converte un RowID interno di Oracle in un tipo di dati NVARCHAR2.

## ROWNUM

**VEDERE ANCHE** PSEUDOCOLONNE, Capitolo 16.

### SINTASSI

ROWNUM

**DESCRIZIONE** ROWNUM restituisce il numero di sequenza con cui è stata restituita una riga la prima volta che è stata selezionata da una tabella. La prima riga ha un valore di ROWNUM pari a 1, la seconda 2 e così via. Si noti che anche un semplice `order by` in un'istruzione `select` può mettere in disordine i ROWNUM, che sono assegnati alle righe prima di qualsiasi operazione di ordinamento.

## RPAD

**VEDERE ANCHE** CARATTERI, FUNZIONI; LPAD; LTRIM; RTRIM; TRIM; Capitolo 7.

### SINTASSI

RPAD(*stringa*, *lunghezza* [, *'set'*])

**DESCRIZIONE** RPAD fa in modo che una stringa raggiunga una certa lunghezza, aggiungendo alla sua destra un determinato *set* di caratteri. Se non si specifica *set*, il carattere di riempimento predefinito è uno spazio.

### ESEMPIO

select RPAD('HELLO ',24,'WORLD') from DUAL;

produce:

HELLO WORLDWORLDWORLDWOR

## RTRIM

**VEDERE ANCHE** CARATTERI, FUNZIONI; LPAD; LTRIM; RPAD; TRIM; Capitolo 7.

### SINTASSI

RTRIM(*stringa* [, *'set'*])

**DESCRIZIONE** RTRIM elimina tutte le occorrenze di un *set* di caratteri a partire dal lato destro di una stringa.

### ESEMPIO

RTRIM('GEORGE','OGRE')

non produce *nulla*, una stringa vuota di lunghezza zero! Invece:

---

RTRIM('EDYTHE', 'HET')

produce:

EDY

## RUN

**VEDERE ANCHE** /, @, @@, EDIT, START.

**SINTASSI**

R[UN]

**DESCRIZIONE** RUN visualizza il comando SQL nel buffer SQL e successivamente lo esegue. RUN è simile al comando /, l'unica differenza è che / non visualizza per primo il comando SQL.

## RUOLO

Un ruolo è un insieme di privilegi che un utente può concedere ad altri utenti. Oracle dispone di diversi ruoli forniti dal sistema, tra cui CONNECT, RESOURCE e DBA. È possibile verificare i privilegi assegnati ai ruoli tramite le viste del dizionario dati ROLE\_SYS\_PRIVS e DBA\_SYS\_PRIVS. Durante lo sviluppo di un'applicazione, non si dovrebbe fare affidamento sui ruoli definiti dal sistema, poiché i nomi e i privilegi di questi ruoli potrebbero cambiare nelle future versioni di Oracle. Per la sintassi utilizzata nella creazione di un ruolo si consulti CREATE ROLE.

Per esempi riguardanti la creazione e l'uso dei ruoli, si *consulti* il Capitolo 19.

## SAVE

**VEDERE ANCHE** EDIT, GET, Capitolo 6.

**SINTASSI**

SAV[E] *file[.est]* [ CRE[ATE] | REP[LACE] | APP[END] ]

**DESCRIZIONE** SAVE salva il contenuto del buffer corrente in un file host di nome *file*. Se non si specifica alcuna estensione, viene utilizzata l'estensione SQL. Questa estensione predefinita può essere modificata con SET SUFFIX. SAVE salva eventuali modifiche ancora pendenti nel database.

Se il file esiste già, l'opzione CREATE causerà l'annullamento dell'operazione di salvataggio e produrrà un messaggio d'errore. L'opzione REPLACE sostituisce qualsiasi file preesistente e ne crea di nuovi all'occorrenza.

L'opzione APPEND aggiunge questo file alla fine di qualunque file già esistente o ne crea uno nuovo se non esiste nessuno.

## ESEMPIO

La riga seguente salva il contenuto del buffer corrente in un file di nome lowcost.sql:

save lowcost

## SAVEPOINT

**VEDERE ANCHE** COMMIT, ROLLBACK, SET TRANSACTION.

### SINTASSI

SAVEPOINT *savepoint*:

**DESCRIZIONE** SAVEPOINT è un punto interno a una transazione da cui è possibile eseguire un rollback. I savepoint consentono di eseguire il rollback di alcune parti della transazione corrente. Vedere ROLLBACK.

La prima versione proposta di seguito è tipica in SQL\*PLUS. Lo scopo di SAVEPOINT è assegnare un nome all'inizio di un gruppo di istruzioni SQL e dopo, se necessario, annullare quell'assegnamento eliminando così solo i risultati *di quel gruppo*. Le istruzioni possono essere annidate o ordinate in sequenza, consentendo un ampio controllo nella gestione degli errori. L'idea di base è raccogliere una serie di istruzioni SQL che costituiscono una transazione logica, ed eseguire il COMMIT solo quando tutte le fasi sono state eseguite con successo. Se un passaggio non riesce, lo si potrà ripetere senza dover disfare nulla. Ecco un esempio:

```
insert into COMFORT
values ('WALPOLE', '22-APR-01', 50.1, 24.8, 0);

savepoint A;

insert into COMFORT
values ('WALPOLE', '27-MAY-01', 63.7, 33.8, 0);

savepoint B;

insert into COMFORT
values ('WALPOLE', '07-AUG-01', 83.0, 43.8, 0);
```

Si può annullare solo l'ultimo inserimento semplicemente tornando al savepoint B:

rollback to savepoint B;

Questo mostra come gruppi di istruzioni SQL possano essere raggruppati e annullati insieme o singolarmente. In un programma host, oppure con PL/SQL o SQL\*PLUS, è possibile configurare dei SAVEPOINT da cui eseguire il ROLLBACK, in base ai risultati dell'ultima istruzione SQL o dell'ultima condizione controllata. ROLLBACK permette di ritornare a un passaggio specifico in cui è stato nominato un SAVEPOINT.

Se una funzione, un controllo logico o un'istruzione SQL ha dei problemi, si possono ripristinare i dati del database allo stato precedente. Quindi si può tentare nuovamente.

I nomi dei SAVEPOINT devono essere unici all'interno di una transazione (che termina dopo un COMMIT o un ROLLBACK incondizionato) ma non necessariamente in tutto il database. Per esempio, si può assegnare a un SAVEPOINT lo stesso nome di una tabella (operazione che potrebbe rendere il codice più comprensibile se al SAVEPOINT si assegnasse lo stesso nome della tabella aggiornata). I nomi dei SAVEPOINT seguono le solite regole di denominazione degli oggetti.

Se a un nuovo SAVEPOINT si assegna un nome già utilizzato, quello nuovo sostituisce quello vecchio. Il vecchio nome viene perso definitivamente e non c'è alcun modo di effettuare un rollback da quel punto.

Una volta effettuato un COMMIT o un ROLLBACK incondizionato (non da un SAVEPOINT), tutti i SAVEPOINT precedenti vengono cancellati. Si ricordi che tutte le istruzioni DDL (come DISCONNECT, CREATE TABLE, CREATE INDEX e così via) eseguono automaticamente un COMMIT implicito e qualsiasi malfunzionamento grave (il programma termina in modo anomalo o il computer si spegne improvvisamente) genera un ROLLBACK automatico.

## **SCAN (SQL\*PLUS)**

*Vedere SET.*

## **SCORE**

La funzione SCORE viene utilizzata dagli indici CONTEXT in Oracle Text per valutare in che misura una stringa di testo rispetti i criteri di ricerca specificati. È possibile visualizzare il “punteggio” di una ricerca individuale; più spesso il punteggio viene confrontato con un valore di soglia. *Vedere il Capitolo 24.*

## **SEGMENTO**

Un segmento è la memoria fisica dello spazio allocato in una tabella, indice o cluster. Una tabella possiede un segmento costituito da tutti i suoi extent. Per ogni indice viene definito un segmento analogo. Un cluster ha almeno due segmenti, uno per i dati e uno per l’indice della chiave di clusterizzazione.

## **SEGMENTO DI ROLLBACK**

Un segmento di rollback è un’area di memoria di una tablespace che contiene le informazioni relative alle transazioni utilizzate per garantire l’integrità dei dati durante un rollback e per fornire coerenza di lettura in più transazioni. *Vedere il Capitolo 40.*

## **SEGMENTO DI UNDO**

A partire da Oracle9i, si potrà ricorrere alla gestione automatica degli undo per fornire coerenza di lettura al posto dei segmenti di rollback. Se si utilizza questa opzione, Oracle gestirà i segmenti di undo per tenere traccia delle modifiche e fornire le capacità di rollback. *Vedere SEGMENTI DI ROLLBACK e il Capitolo 40.*

## **SEGMENTO TEMPORANEO**

Si dice segmento temporaneo l’area di memoria all’interno di una tablespace utilizzata per contenere i risultati intermedi di un’istruzione SQL. Per esempio, i segmenti temporanei vengono utilizzati nel riordino di tabelle molto estese. Solitamente, i segmenti temporanei sono memorizzati in tablespace a essi dedicate; *vedere CREATE TABLESPACE.*

## SELECT (Forma 1, SQL)

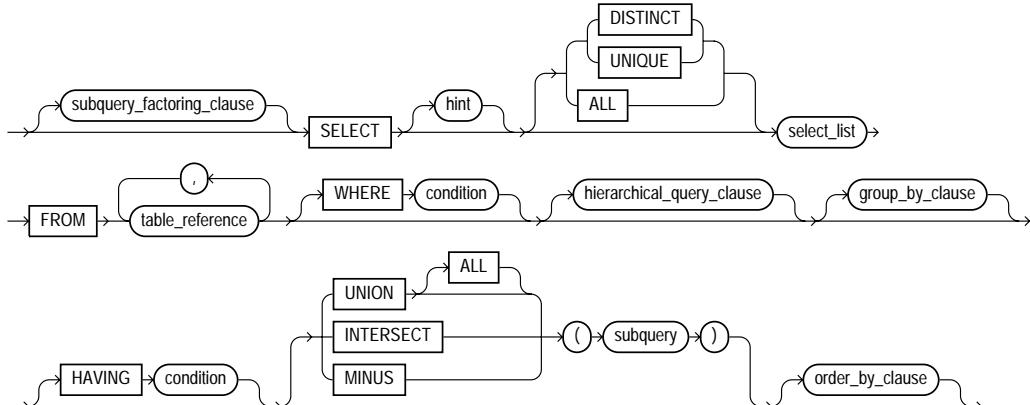
VEDERE ANCHE COLLATION, CONNECT BY, DELETE, DUAL, FROM, GROUP BY, HAVING, INSERT, JOIN, OPERATORI LOGICI, ORDER BY, OPERATORI DI QUERY, SUBQUERY, OPERATORI SINTATTICI, UPDATE, WHERE.

### SINTASSI

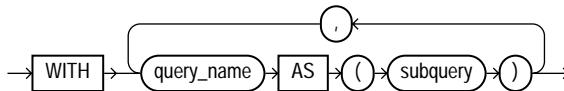
select



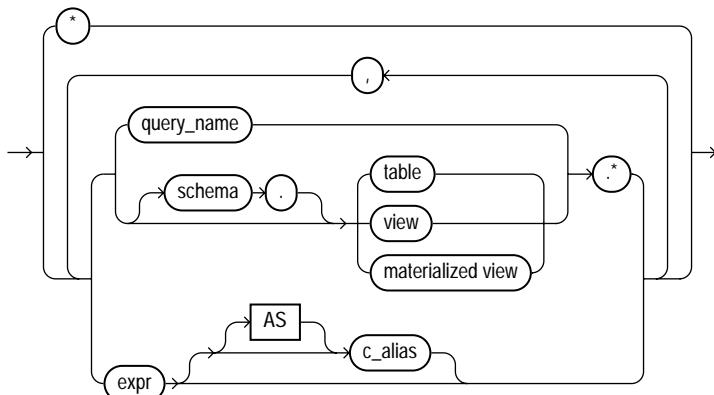
subquery

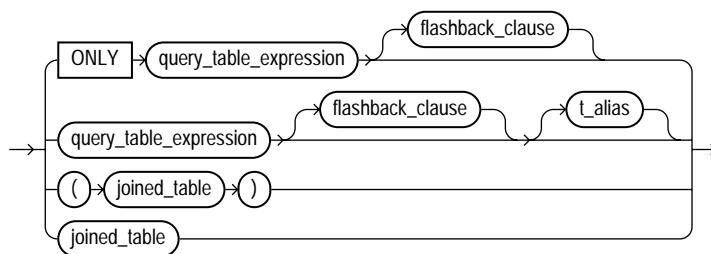
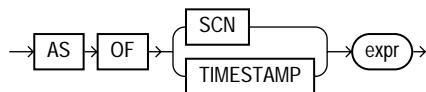
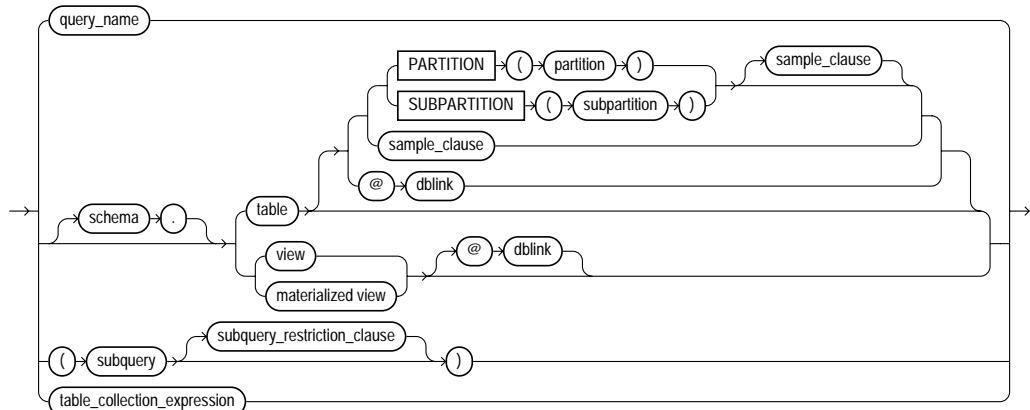
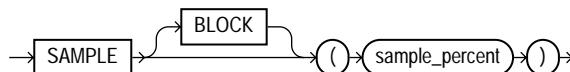
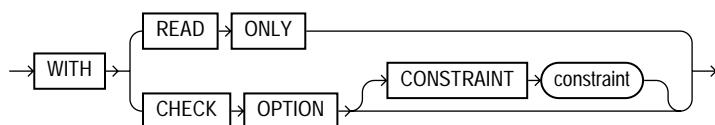


subquery\_factoring\_clause

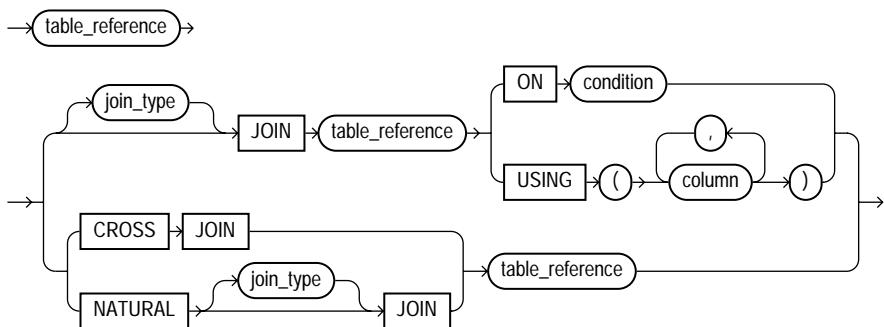


select\_list

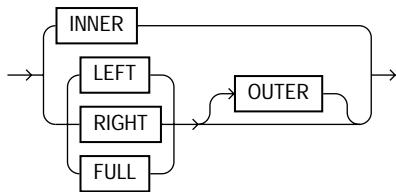


**table\_reference****flashback\_clause****query\_table\_expression****sample\_clause****subquery\_restriction\_clause****table\_collection\_expression**

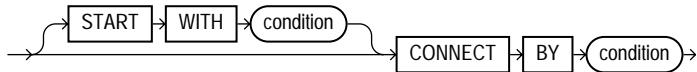
## joined\_table



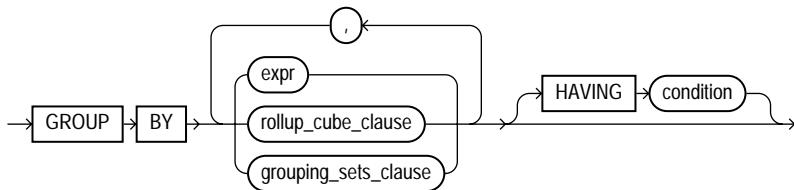
**join\_type**



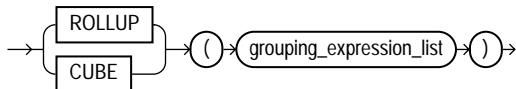
## **hierarchical\_query\_clause**



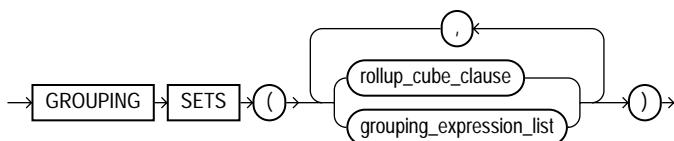
## group\_by\_clause

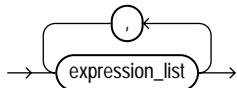
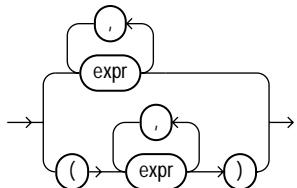
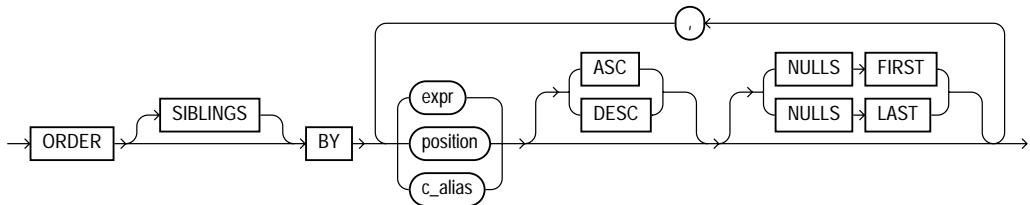
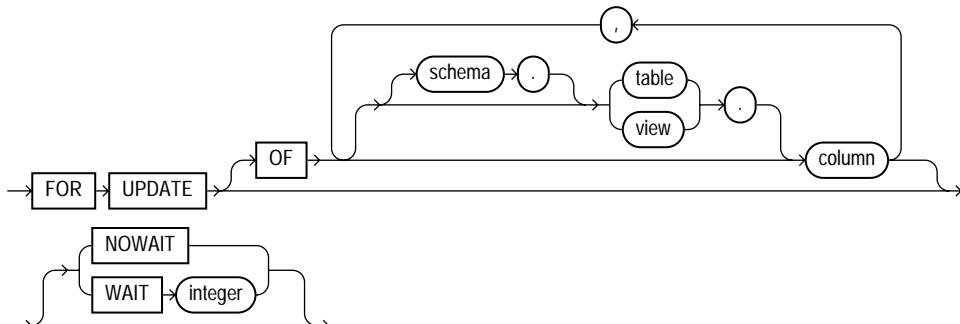


## rollup\_cube\_clause



## grouping\_sets\_clause



**grouping\_expression\_list****expression\_list****order\_by\_clause****for\_update\_clause**

**DESCRIZIONE** **SELECT** recupera delle righe da una o più tabelle o viste, come comando a sé stante o come subquery all'interno di altri comandi SQL (con limitazioni) compresi select, insert, update e delete. **ALL** impone che vengano restituite tutte le righe che soddisfano le condizioni specificate (ed è l'impostazione predefinita). **DISTINCT** impone che vengano restituite solo le righe uniche, gli eventuali doppioni vengono ignorati.

Un \* (asterisco) da solo comporta la visualizzazione di tutte le colonne prese da tutte le tabelle della clausola from. *tabella.\** significa che verranno visualizzate tutte le colonne di questa tabella. Un asterisco da solo non può essere combinato con qualsiasi altra colonna, mentre *tabella.\** può essere seguita da nomi di colonna e espressioni, tra cui ROWID.

**select\_list** può includere qualsiasi tipo di colonna. Può trattarsi del nome di una colonna, di un letterale, di un calcolo matematico, di una funzione o di una combinazione di funzioni. La

versione più comune è un semplice nome di colonna, eventualmente preceduto dal nome di una tabella o da un suo alias. *alias\_colonna* è l'alias di una colonna o di un'espressione. L'alias di colonna può essere preceduto dalla parola AS. L'alias diviene l'intestazione di colonna nella visualizzazione e può essere indicato mediante i comandi SQL\*PLUS column, ttitle e btitle. Può inoltre essere utilizzato nella clausola order by. Se è composto da più di una parola oppure contiene caratteri che non sono né numeri né lettere, deve essere racchiuso tra virgolette.

*schema*, *tabella* e *dblink* denotano la tabella o la vista da cui prelevare le righe; *schema* e *dblink* sono facoltativi, la loro assenza fa in modo che Oracle supponga si tratti dell'utente corrente. Specificando il nome utente in una query si può ridurre il carico di lavoro di Oracle e eseguire la query più velocemente. Nella clausola FROM, si può specificare una subquery. Oracle esegue la subquery e le righe risultanti vengono trattate come se provenissero da una vista.

*alias\_tabella* in questo caso rinomina la tabella solo per la query corrente. A differenza degli alias delle espressioni, a questo alias si può fare riferimento da qualsiasi altro punto dell'istruzione select, per esempio come prefisso nei nomi di colonne provenienti da questa tabella. Se una tabella dispone di un alias, questo deve essere utilizzato in ogni prefisso di colonna che lo richiede. Alle tabelle vengono associati degli alias generalmente quando due tabelle con nomi lunghi vengono unite tramite join in una query, e quando molte delle loro colonne hanno nomi identici, rendendo necessaria una denominazione chiara nella clausola select. Inoltre, gli alias vengono utilizzati spesso in subquery correlate e quando una tabella viene unita tramite join a se stessa.

*condizione* può essere qualsiasi espressione valida che vale vero o falso. Può contenere funzioni, colonne (prese da queste tabelle) e letterali. *posizione* consente alla clausola order by di identificare le espressioni in base alla loro posizione relativa nella clausola select piuttosto che ai loro nomi. Ciò si rivela particolarmente utile nelle espressioni complesse. Si dovrebbe alternare l'uso delle posizioni ordinali nelle clausole order by; gli alias di colonna possono essere impiegati in terza battuta. Lo standard SQL attuale non prevede alcun supporto per l'uso di posizioni ordinali nelle clausole order by e quindi Oracle potrebbe non supportare tale uso in futuro. ASC e DESC specificano rispettivamente se per ogni espressione nella clausola order by si tratta di una sequenza crescente o decrescente. Per un'analisi di queste sequenze, si consulti COLLATION.

*colonna* è una colonna di una tabella elencata nella clausola from. Non può essere un'espressione, ma deve essere un nome vero di colonna. NOWAIT significa che ogni eventuale select for update che incontra una riga bloccata viene arrestato e il controllo torna immediatamente all'utente invece di aspettare lo sblocco della risorsa e cercare di aggiornare ancora la riga in breve tempo.

La clausola for update of inserisce dei blocchi sulle righe selezionate. select ... for update of dovrebbe essere seguita subito da un comando update...where, oppure da un COMMIT o ROLLBACK se si decide di non eseguire alcun aggiornamento. for update of include anche le operazioni di inserimento e cancellazione. Una volta bloccata una riga, gli altri utenti non potranno aggiornarla fino a quando non la si libererà con un comando COMMIT (o AUTOCOMMIT) o ROLLBACK. select... for update of non può includere DISTINCT, GROUP BY, UNION, INTERSECT, MINUS, o alcuna funzione di gruppo, come MIN, MAX, AVG, o COUNT. Le colonne indicate a questo punto non hanno alcun effetto e sono presenti per mera compatibilità con altri dialetti SQL. Oracle blocca solo le tabelle che appaiono all'interno della clausola for update. Se non si dispone di una clausola of in cui elencare le tabelle, Oracle blocca tutte le tabelle della clausola from per aggiornarle. Tutte le tabelle devono trovarsi nello stesso database e, se sono presenti dei riferimenti a una colonna di tipo LONG o a una sequenza nell'istruzione select, le tabelle devono trovarsi nello stesso database della colonna LONG o della sequenza.

Se si esegue una query da una tabella partizionata è possibile indicare il nome della partizione su cui si esegue la query all'interno della clausola from della query. Vedere il Capitolo 18.

Le altre singole clausole dell'istruzione select sono descritte sotto il loro nome in questo stesso capitolo. *clausola\_flashback* sarà disponibile a partire da Oracle9i, Release 2.

#### **Altre note**

Le clausole devono essere disposte nell'ordine indicato, fatta eccezione per quelle che seguono:

- connect by, start with, group by e having, che possono essere inserite in qualsiasi ordine relativo le une alle altre;
- order by e for update of, che possono essere inserite in qualsiasi ordine l'una rispetto all'altra.

## **SELECT (Forma 2, Embedded SQL)**

**VEDERE ANCHE** CONNECT, DECLARE CURSOR, DECLARE DATABASE, EXECUTE, FETCH, FOR, PREPARE, UPDATE (Forma 1), WHENEVER, guida di programmazione del precompilatore.

#### **SINTASSI**

```
EXEC SQL [AT { db_nome | :variabile_host }] 
  SELECT select_list INTO
    :variabile_host [[INDICATOR] :variabile_indicatore]
    [, :variabile_host [[INDICATOR] :variabile_indicatore]]...
    [FROM elenco_tabella [WHERE condizione]
     { [START WITH condizione] CONNECT BY condizione
      | GROUP BY espr [, espr]... [HAVING condizione] }
     [ [START WITH condizione] CONNECT BY condizione
      | GROUP BY espr [, espr]... [HAVING condizione] ]...
     [ WITH READ ONLY | WITH CHECK OPTION ]
     [{ UNION | UNION ALL | INTERSECT | MINUS } SELECT command]
    [ ORDER BY
      { espr | posizione } [ ASC | DESC ]
      [, { espr | posizione } [ ASC | DESC ]]...
     | FOR UPDATE
       [OF [[schema.] {tabella. | vista . | vista materializzata. }] colonna
        [, OF [[schema.] {tabella. | vista . | vista materializzata. }] colonna]... ] ]
    [NOWAIT]
    [ ORDER BY
      { espr | posizione } [ ASC | DESC ]
      [, { espr | posizione } [ ASC | DESC ]]...
     | FOR UPDATE
       [OF [[schema.] {tabella. | vista . | vista materializzata. }] colonna
        [, OF [[schema.] {tabella. | vista . | vista materializzata. }] colonna]... ] ]
    [NOWAIT]
```

**DESCRIZIONE** Vedere la descrizione delle varie clausole nella forma 1 di SELECT. Di seguito sono elencati gli elementi caratteristici di Embedded SQL:

- AT database, che assegna facoltativamente un nome a un database da una istruzione CONNECT precedente, per un nome di database da una precedente istruzione DECLARE DATABASE.
- INTO :variabile\_host [, variabile\_host] è l'elenco di variabili host in cui dovranno essere scritti i risultati dell'istruzione select. Se una variabile di questa lista è un array, devono esserlo anche tutte le altre anche se non è necessario che siano della stessa dimensione.

- La clausola `where` può riferirsi a variabili host non array.
- La clausola `select` può riferirsi a variabili host quando si può utilizzare una costante.

Si può utilizzare la versione Embedded SQL di `SELECT` con array variabili per restituire più righe con un unico `FETCH`. Inoltre è possibile utilizzare questa versione per dichiarare cursori o preparare istruzioni da usare in un `FETCH` successivo, specificando eventualmente la clausola `for update of`. L'ultima istruzione `update` potrà quindi riferirsi alle colonne indicate nelle clausole `for update of` attraverso la propria clausola `current of`. Vedere `DECLARE CURSOR` e `UPDATE` (Forma 2).

Se questo comando non restituisce alcuna riga, `SQLCODE` varrà +100 e le variabili host manterranno il contenuto che avevano prima dell'esecuzione del comando. In tal caso, `WHENEVER` consente di reindirizzare il flusso di esecuzione.

Tutte le righe che soddisfano i criteri di selezione vengono bloccate alla loro apertura. `COMMIT` rilascia i blocchi e da quel momento non sono consentiti ulteriori `FETCH`. Questo significa che è indispensabile eseguire il `FETCH` ed elaborare tutte le righe da aggiornare prima di impartire il comando `COMMIT`.

## **SELECT...INTO**

**VEDERE ANCHE** `%ROWTYPE`, `%TYPE`, `DECLARE VARIABLE`, `FETCH`.

### **SINTASSI**

```
SELECT espressione [.espressione]...
  INTO {variabile [.variabile]... | record}
  FROM [utente.]tabella [, [utente.]tabella]...
  [where...][group by... [having...]] [order by...];
```

**DESCRIZIONE** Questo non è il formato dell'istruzione `select` impiegato in `DECLARE CURSOR`. Questo formato utilizza il cursore隐式 di nome SQL, viene eseguito all'interno di una sezione di esecuzione di un blocco (racchiuso tra `BEGIN` ed `END`) e copia i valori da un'unica riga restituita sotto forma di stringa di variabili nominate, o di record la cui struttura sia stata dichiarata da `%ROWTYPE` in modo che rispecchi le colonne selezionate. Se viene utilizzata la versione che usa le variabili, queste devono essere state dichiarate e devono avere tipi di dati compatibili con quelle recuperate. L'ordine in cui compaiono nella clausola `into` deve coincidere con l'ordine delle colonne associate presenti nella clausola `select`.

Questo tipo di `select` può essere utilizzato in un ciclo con variabili PL/SQL che appaiano nella clausola `where` (o che si comportano come costanti nella clausola `select`), ma ogni volta che si esegue `select` questa deve restituire una sola riga. Se al contrario ne restituisce più di una, viene sollevata l'eccezione `TOO_MANY_ROWS` e `SQL%ROWCOUNT` varrà 2 (per ulteriori dettagli vedere ECCEZIONI). `SQL%FOUND` varrà TRUE.

Se non restituisce alcuna riga, viene sollevata l'eccezione `NO_DATA_FOUND`, `SQL%ROWCOUNT` viene impostato a 0 e `SQL%FOUND` a FALSE.

## **SEQUENZA**

Una sequenza è un oggetto di database utilizzato per generare numeri interi unici da impiegare come chiavi primarie. Vedere `CREATE SEQUENCE`.

## **SERVER CONDIVISO**

Il server condiviso di Oracle, noto in precedenza come Multi-Threaded Server (MTS), supporta connessioni a Oracle tramite dispatcher process e processi a server condiviso. I dispatcher process accettano richieste di connessione da parte degli utenti e i processi a server condiviso comunicano con il database. Il server condiviso permette a molti utenti di condividere un piccolo numero di processi a server condiviso, riducendo così la quantità di memoria richiesta per supportare gli utenti del database. Il server condiviso è particolarmente efficiente quando ci sono più utenti che avviano connessioni e quando gli utenti delle applicazioni interrompono spesso le proprie attività di inserimento dei dati.

## **SERVER MULTITHREAD**

*Vedere SERVER CONDIVISO.*

## **SESSIONE**

Una sessione è quella serie di eventi che si verificano dal momento in cui un utente si connette a SQL al momento in cui si disconnette.

## **SESSIONTIMEZONE**

**VEDERE ANCHE** Capitolo 8.

### **SINTASSI**

SESSIONTIMEZONE

**DESCRIZIONE** SESSIONTIMEZONE restituisce il valore del fuso orario della sessione corrente.

### **ESEMPIO**

```
select SESSIONTIMEZONE from DUAL;
```

## **SET**

**VEDERE ANCHE** SET TRANSACTION, SHOW, Capitolo 6.

### **SINTASSI**

SET *caratteristica* *valore*

dove *caratteristica* e *valore* appartengono all'elenco seguente. I valori predefiniti sono sottolineati.

APPI[NFO] {ON|OFF|testo} abilita la registrazione di file dei comandi attraverso il package DBMS\_APPLICATION\_INFO. Il nome registrato compare nella colonna Module della tabella di prestazioni dinamiche V\$SESSION.

ARRAY[SIZE] {15|n} imposta la dimensione del batch di righe che SQL\*PLUS recupera a ogni prelievo. L'intervallo spazia da 1 a 5000. Valori più grandi migliorano l'efficienza delle query e delle subquery in cui ha senso recuperare molte righe ma viene utilizzata molta memoria. Valori superiori a 100 raramente producono miglioramenti sensibili a livello di prestazioni. ARRAYSIZE non ha alcun altro effetto su SQL\*PLUS.

AUTO[COMMIT] {ON|OFF|IMMEDIATE}|n fa in modo che SQL salvi immediatamente qualsiasi modifica al database non appena ogni comando SQL viene completato. n consente di impostare il numero di comandi dopo cui effettuare il salvataggio. OFF interrompe il salvataggio automatico obbligando, di fatto, a indicare esplicitamente le richieste di salvataggio tramite il comando COMMIT. Molti comandi, come QUIT, EXIT, CONNECT e tutti i comandi DDL eseguono essi stessi un COMMIT di qualsiasi modifica ancora pendente.

AUTOP[RINT] {ON|OFF} decide se SQL\*Plus debba visualizzare automaticamente le bind variable utilizzate in un blocco PL/SQL o in una procedura eseguita.

AUTORECOVERY [ON|OFF] indica al comando RECOVER di applicare automaticamente i nomi di file predefiniti dei redo log file archiviati che si rendono necessari durante le operazioni di recupero.

AUTOT[RACE] {ON|OFF|TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]] permette di visualizzare il percorso esecutivo di una query dopo che si è conclusa. Per visualizzare il percorso esecutivo, per prima cosa si deve creare la tabella PLAN\_TABLE nel proprio schema (può essere creata eseguendo il file UTLXPLAN.SQL all'interno della directory /rdbms/admin che si trova nella directory del software Oracle).

BLO[CKTERMINATOR] {.|c} imposta il simbolo usato per indicare la fine di un blocco PL/SQL. Non può essere né una lettera né un numero. Per eseguire il blocco, occorre utilizzare i comandi RUN o / (barra).

CMDS[EP] {;|c|ON|OFF} imposta il carattere usato per separare più comandi SQL\*PLUS impartiti eventualmente su un'unica linea. ON e OFF decidono se consentire o meno la presenza di più comandi su un'unica linea.

COLSEP {\_|testo} imposta un valore da visualizzare tra le colonne.

COMPATIBILITY {V7|V8|NATIVE} specifica la versione di Oracle a cui si è connessi.

CON[CAT] {.|c|ON|OFF} imposta un simbolo che può essere impiegato per terminare o delimitare una variabile utente seguita da un simbolo, un carattere o una parola che altrimenti potrebbe essere interpretata come parte del nome della variabile. Impostando CONCAT a ON il suo valore torna a ‘.’.

COPYC[OMMIT] {0|n} salva le righe nel database di destinazione su un ciclo di n batch di righe (il numero di righe in un batch è impostato da ARRAYSIZE). I valori possibili variano tra 0 e 5000. Se valore vale 0, il COMMIT viene eseguito solo al termine dell'operazione di copia.

COPYTYPECHECK {ON|OFF} permette di eliminare i controlli di tipo mentre si usa il comando COPY.

DEF[INE] {&|c|ON|OFF} decide se SQL\*PLUS debba cercare e sostituire i comandi delle variabili di sostituzione e se debba caricarle insieme ai loro valori definiti.

DESCRIBE [DEPTH {1|n|ALL}][LINENUM {ON|OFF}][INDENT {ON|OFF}] imposta la profondità del livello fino a dove è possibile descrivere in maniera ricorsiva un oggetto. L'intervallo valido della clausola DEPTH va da 1 a 50.

ECHO {ON|OFF} ECHO ON fa in modo che SQL\*PLUS ripeta (visualizzi) i comandi sullo schermo mentre vengono eseguiti da un file di avvio. Con OFF, SQL\*PLUS esegue i comandi senza visualizzarli. L'output dei comandi è controllato da TERMOUT.

EDITF[ILE] nome\_file[.ext] imposta il nome di file predefinito creato dal comando EDIT.

EMB[EDDED] {ON|OFF} consente a un nuovo report di una serie di incominciare in qualsiasi punto di una pagina, subito dopo la conclusione di quello precedente. OFF induce il nuovo report a incominciare all'inizio di una nuova pagina.

ESC[APE] {\|c|ON|OFF} definisce il carattere inserito come carattere di escape. OFF non definisce il carattere di escape. ON attiva il carattere di escape. ON imposta nuovamente il valore di c a quello predefinito, “\”. Il carattere di escape può essere utilizzato prima di quello di sostituzione (impostato tramite SET DEFINE) per indicare che SQL\*Plus dovrebbe trattare il carattere di sostituzione come un carattere ordinario e non come una richiesta di sostituire la variabile.

**FEED[BACK] {6|n|ON|OFF}** fa in modo che SQL\*PLUS visualizzi i “record selezionati” dopo una query se sono stati selezionati almeno *n* record. ON e OFF abilitano e disabilitano questa funzione. SET FEEDBACK 0 ha lo stesso effetto di OFF.

**FLAGGER {OFF|ENTRY |INTERMED|IATE||FULL}** imposta il livello FIPS per la compatibilità SQL92.

**FLU[SH] {ON|OFF}** viene utilizzato quando un file di avvio può essere eseguito senza richiedere la visualizzazione di alcun messaggio o alcun intervento dell’utente fino al suo completamento. OFF permette al sistema operativo di evitare l’invio dell’output sullo schermo. ON riabilita l’output sul terminale. OFF può in taluni casi migliorare le prestazioni.

**HEA[DING] {ON|OFF}** elimina le intestazioni, testo e caratteri di sottolineatura, che normalmente appaiono sopra le colonne. ON ne riabilita la visualizzazione.

**HEADS[EP] {||c|ON|OFF}** definisce il carattere inserito come carattere separato dell’intestazione. ON e OFF rispettivamente attivano e disattivano la separazione dell’intestazione.

**INSTANCE [percorso\_istanza |LOCAL]** imposta l’istanza predefinita per la sessione al percorso di istanza specificato. L’uso del comando SET INSTANCE non comporta la connessione a un database. L’istanza predefinita viene utilizzata per i comandi quando non si specifica nessun altra istanza.

**LIN[ESIZE] {80|n}** imposta la larghezza della linea, il numero totale di caratteri visualizzabili su una linea prima di passare a quella successiva. Questo numero viene utilizzato anche quando SQL\*PLUS calcola la posizione corretta dei titoli centrati o allineati a destra. Il valore massimo di *n* è 999.

**LOBOF[FSET] {n|1}** imposta la posizione iniziale da cui vengono recuperati e visualizzati i dati CLOB e NCLOB.

**LOGSOURCE [nome\_percorso]** specifica la posizione da cui vengono recuperati i log archiviati durante il recupero. Il valore predefinito è impostato dal parametro di inizializzazione LOG\_ARCHIVE\_DEST che si trova nel file di inizializzazione di Oracle. L’uso del comando SET LOGSOURCE senza un nome di percorso ripristina la locazione predefinita.

**LONG {80|n}** è la larghezza massima per visualizzare o copiare (spooling) i valori LONG, CLOB e NCLOB. *n* può variare tra 1 e 2 GB.

**LONGC[HUNKSIZE] {80|n}** imposta la dimensione, in caratteri, degli incrementi in cui SQL\*Plus recupera un valore LONG, CLOB o NCLOB.

**MARK[UP] HTML [ON|OFF] [HEAD testo] [BODY testo] [TABLE testo] [ENTMAP {ON|OFF}] [SPOOL {ON|OFF}] [PRE[FORMAT] {ON|OFF}]** visualizza il testo con markup HTML. SET MARKUP specifica solamente che l’output di SQL\*Plus verrà codificato con HTML. Si deve utilizzare SET MARKUP HTML ON SPOOL ON e il comando SQL\*Plus SPOOL per creare e nominare un file di spool e per iniziare a scrivere l’output HTML in questo file. SET MARKUP ha le stesse opzioni e il medesimo comportamento di SQLPLUS -MARKUP.

**NEWP[AGE] {1|n|NONE}** imposta il numero di linee vuote da visualizzare tra il fondo di una pagina e il titolo iniziale di quella seguente. Uno 0 (zero) invia un avanzamento carta (form feed) all’inizio di ogni pagina. Se l’output viene visualizzato, generalmente questo corrisponde alla cancellazione dello schermo.

**NULL testo** imposta il testo che SQL\*PLUS sostituisce quando scopre valori NULL. NULL senza testo visualizza degli spazi vuoti.

**NUMF[ORMAT]** formato imposta il formato numerico predefinito per la visualizzazione di dati numerici. Per dettagli, si consulti NUMERICI, FORMATI.

**NUM[WIDTH] {10|n}** imposta la larghezza predefinita per la visualizzazione dei numeri.

**PAGES[IZE] {24|n}** imposta il numero di linee per pagina.

**PAU[SE] {ON|OFF|testo}** ON impone a SQL\*PLUS di attendere che l’utente prema il tasto INVIO dopo la visualizzazione di ogni pagina di output. OFF disabilita questa funzione. *testo* è il

messaggio che SQL\*PLUS visualizza in fondo allo schermo mentre attende che l'utente prema il tasto INVIO. Anche per visualizzare la prima pagina è necessario premere INVIO.

**RECSEP {WR[APPED]|EA[CH]|OFF}**: Ogni riga restituita da SQL\*PLUS può essere separata dalle altre tramite un carattere, definito da RECSEPCHAR (*vedere anche SET*). Per default, questo carattere è uno spazio e la funzione è attiva sui soli record (e quindi sulle colonne) che vanno a capo.

RECSEPCHAR funziona con RECSEP. Il simbolo predefinito è uno spazio, ma lo si può impostare al simbolo che si desidera.

**SERVEROUT[PUT] {ON|OFF} [SIZE n] [FOR[MAT] {WRA[PPED] | WORD\_WWRAPPED}| TRU[NCATED]}** consente di visualizzare l'output delle procedure PL/SQL (dal package DBMS\_OUTPUT, *vedere Application Developer's Guide*). WRAPPED, WORD\_WWRAPPED, TRUNCATED e FORMAT determinano il tipo di formattazione del testo di output.

**SHIFT[INOUT] {VIS[IBLE]|INV[ISIBLE]}** consente il corretto allineamento dei terminali che visualizzano i caratteri maiuscoli. Si utilizzi VISIBLE per i terminali che visualizzano i caratteri maiuscoli come caratteri visibili (per esempio, uno spazio o una colonna). INVISIBLE è esattamente l'opposto e non visualizza caratteri maiuscoli.

**SHOW[MODE] {ON|OFF}** ON fa in modo che SQL\*PLUS visualizzi le impostazioni precedenti e quelle successive di un comando SET, e i suoi valori quando viene modificato. OFF disattiva entrambe.

**SQLBL[ANKLINES] {ON|OFF}** controlla se SQL\*Plus ammette la presenza di linee vuote all'interno di un comando o di uno script SQL. ON interpreta le linee vuote e quelle nuove come parte di un comando o di uno script SQL. OFF, il valore predefinito, non ammette la presenza di linee vuote o nuove in un comando o in uno script SQL.

**SQLC[ASE] {MIX[ED]|LO[WER]|UP[PER]}** converte tutto il testo, compresi letterali e identificatori contenuti in blocchi SQL o PL/SQL, in lettere maiuscole o minuscole prima di farlo eseguire da Oracle. MIXED non altera i caratteri maiuscoli e minuscoli. LOWER converte ogni cosa in lettere minuscole; UPPER converte ogni cosa in lettere maiuscole.

**SQLCO[NTINUE] {>|testo}** imposta la sequenza di caratteri da visualizzare come messaggio guida quando una linea troppo lunga deve continuare in quella successiva. Se si inserisce un trattino (-) alla fine di una linea di comando di SQL\*PLUS, i simboli SQLCONTINUE o il testo visualizzano il prompt sulla sinistra della linea seguente.

**SQLN[UMBER] {ON|OFF}**: Se si attiva questa opzione, le linee SQL che si trovano oltre quella inserita per prima avranno numeri di linea come messaggi guida. Se la si disattiva, SQLPROMPT apparirà solo sulle linee in più.

**SQLPLUSCOMPAT[IBILITY] {x.y[.z]}** imposta il comportamento o il formato di output di VARIABLE a quello della release o della versione specificata da x.y[.z]. In cui x è il numero di *versione*, y è il numero della *release* mentre z è il numero dell'*aggiornamento*. Nelle release successive, SQLPLUSCOMPATIBILITY potrà influire su caratteristiche diverse da VARIABLE. Se si imposta il valore di SQLPLUSCOMPATIBILITY a una versione precedente a 9.0.0, la definizione VARIABLE dei tipi di dati NCHAR o NVARCHAR2 ritornerà al comportamento di Oracle8i con il quale la dimensione della variabile è espressa in byte o in caratteri in base al set di caratteri nazionali scelto.

**SQLPRE[FIX] {#|c}** imposta il carattere del prefisso di SQL\*Plus. Mentre si inserisce un comando SQL o un blocco PL/SQL, si può inserire anche un comando SQL\*Plus su una linea separata, preceduta dal carattere del prefisso di SQL\*Plus. SQL\*Plus eseguirà il comando immediatamente senza influire sul comando SQL o sul blocco PL/SQL che si inserisce. Il carattere del prefisso non deve essere un carattere alfanumerico.

**SQLP[ROMPT] {SQL>}|testo** imposta il prompt dei comandi SQL\*Plus.

**SQL[TERMINATOR] {;|c|ON|OFF}** imposta al valore c il carattere utilizzato per terminare e eseguire i comandi SQL.

**SUF[FIX] {SQL|testo}** imposta l'estensione di file predefinita che SQL\*Plus usa nei comandi che fanno riferimento ai file dei comandi. SUFFIX non controlla le estensioni per i file di spool.

**TAB {ON|OFF}** determina in che modo SQL\*Plus formatta gli spazi vuoti nell'output del terminale. OFF usa gli spazi per formattare gli spazi vuoti nell'output. ON usa il carattere TAB. Il carattere TAB viene impostato ogni otto caratteri.

**TERM[OUT] {ON|OFF}**: OFF elimina la visualizzazione in modo che sia possibile eseguire lo spooling dell'output da un file dei comandi senza visualizzare l'output sullo schermo. ON visualizza l'output.

**TI[ME] {ON|OFF}** ON visualizza l'ora corrente prima di ogni prompt dei comandi. OFF disattiva la visualizzazione dell'ora.

**TIM[ING] {ON|OFF}** ON visualizza le statistiche di sincronizzazione per ciascun comando SQL o blocco PL/SQL eseguito. OFF elimina la sincronizzazione di ogni comando.

**TRIM[OUT] {ON|OFF}** ON rimuove gli spazi vuoti alla fine di ogni linea, migliorando le prestazioni soprattutto quando si accede a SQL\*Plus da un dispositivo di comunicazione lento. OFF consente a SQL\*Plus di visualizzare gli spazi vuoti finali.

**TRIMS[POOL] {ON|OFF}** ON rimuove gli spazi vuoti alla fine di ogni linea. OFF consente a SQL\*Plus di includere gli spazi vuoti finali.

**UND[ERLINE] {-|c|ON|OFF}** imposta il carattere impiegato per sottolineare le intestazioni di colonna nei report SQL\*Plus.

**VER[IFY] {ON|OFF}** controlla se SQL\*Plus elenca il testo di un'istruzione SQL o di un comando PL/SQL prima e dopo che aver sostituito le variabili di sostituzione con dei valori. ON elenca il testo; OFF elimina il listato.

**WRA[P] {ON|OFF}** controlla se SQL\*Plus tronca la visualizzazione di una riga selezionata nel caso in cui questa sia troppo lunga per la larghezza della linea corrente. OFF tronca la riga selezionata; ON consente alla riga selezionata di continuare nella linea successiva.

**DESCRIZIONE** SET imposta una caratteristica di SQL\*PLUS, attivandola, disattivandola o assegnandole un certo valore. Tutte queste caratteristiche sono modificate con il comando SET e visualizzate tramite il comando SHOW. SHOW visualizza un comando in particolare se il suo nome segue la parola SHOW, o tutti i comandi se la parola SHOW viene inserita da sola.

## SET CONDITION

La condizione di impostazione (set condition) è un'espressione logica contenente una query, per esempio "Nome IN (select...)". Il suo nome deriva dal fatto che la condizione di query produrrà un gruppo di record.

## SET CONSTRAINTS

**VEDERE ANCHE** VINCOLO DI INTEGRITÀ.

### SINTASSI

```
SET { CONSTRAINT | CONSTRAINTS }
{ vincolo [, vincolo]... | ALL }
{ IMMEDIATE | DEFERRED }
```

**DESCRIZIONE** SET CONSTRAINTS DEFERRED comunica a Oracle di non controllare la validità di un vincolo fino a quando sulla transazione non sarà stato eseguito il commit. In questo modo, si può differire momentaneamente il controllo dell'integrità dei dati. Solitamente, i vincoli differiti vengono utilizzati quando si esegue il DML su più tabelle correlate e non è possibile garantire l'ordine delle transazioni. Per eseguire queste operazioni si deve disporre del privilegio SELECT sulla tabella oppure essere i legittimi proprietari della tabella modificata dal vincolo.

L'impostazione predefinita è SET CONSTRAINTS IMMEDIATE: i controlli di integrità vengono effettuati non appena si inserisce un nuovo record nella tabella.

## SET ROLE

**VEDERE ANCHE** ALTER USER, CREATE ROLE, Capitolo 19.

### SINTASSI

```
SET ROLE
{ ruolo [IDENTIFIED BY password] [, ruolo [IDENTIFIED BY password]]...
| ALL [EXCEPT ruolo [, ruolo] ...]
| NONE};
```

**DESCRIZIONE** SET ROLE abilita o disabilita i ruoli concessi a un utente nella sessione SQL corrente. La prima opzione consente di specificare dei ruoli fornendo facoltativamente una password, se il ruolo lo richiede, (*vedere* CREATE ROLE). La seconda opzione permette di abilitare tutti (ALL) i ruoli a eccezione (EXCEPT) di quelli indicati, che dovranno essere concessi direttamente all'utente, e non tramite altri utenti. L'opzione ALL non abilita i ruoli protetti da password. La terza opzione, NONE, disabilita tutti i ruoli della sessione corrente.

## SET TRANSACTION

**VEDERE ANCHE** COMMIT, ROLLBACK, SAVEPOINT, TRANSACTION, Capitolo 40.

### SINTASSI

```
SET TRANSACTION
{ { READ { ONLY | WRITE }
| ISOLATION LEVEL { SERIALIZABLE | READ COMMITTED }
| USE ROLLBACK SEGMENT segmento_rollback }
[NAME 'test']
| NAME 'test' } ;
```

**DESCRIZIONE** SET TRANSACTION avvia una transazione. La transazione SQL standard garantisce la coerenza di lettura a livello di istruzione, assicurando così che i dati di una query siano coerenti durante l'esecuzione dell'istruzione stessa. Tuttavia, alcune transazioni richiedono garanzie maggiori dato che una serie di istruzioni select, ciascuna all'opera su una o più tabelle, dovranno visualizzare dati coerenti gli uni con gli altri (ovvero dati che si riferiscono a uno stesso istante di tempo). SET TRANSACTION READ ONLY specifica questa coerenza di lettura più forte. Nessuna delle modifiche che altri utenti potrebbero apportare ai dati su cui sono state eseguite query influirà sulla vista che la transazione ha dei dati stessi. In una transazione di sola lettura è possibile impiegare solo i comandi select, lock table, set role, alter session e alter system.

Alle transazioni che contengono i comandi `insert`, `update` o `delete`, Oracle assegna un segmento di rollback. `SET TRANSACTION USE ROLLBACK SEGMENT` specifica che la transazione corrente utilizzerà un segmento di rollback specifico.

`SET TRANSACTION` dev'essere la prima istruzione SQL di una transazione. L'inserimento di un `COMMIT` subito prima di `SET TRANSACTION` assicura la veridicità di questa impostazione. Il `COMMIT` alla fine rilascia la vista materializzata dei dati. Questo è importante perché Oracle ha bisogno di rilasciare le risorse che ha messo da parte per mantenere la coerenza.

## SGA

Vedere System Global Area.

## SHARED POOL

Un'area nella SGA di Oracle che contiene sia la cache del dizionario dati sia un'area condivisa per le versioni analizzate di tutti i comandi SQL del database. La sua dimensione è stabilita dal parametro `SHARED_POOL_SIZE` che si trova nel file dei parametri di inizializzazione del database.

## SHOW

**VEDERE ANCHE** SET.

### SINTASSI

`SHO[W] { opzione }`

*opzione* può essere qualsiasi parametro impostato tramite il comando SET o la sintassi seguente:

```
variabile_sistema
ALL
BTI[TLE]
ERR[ORS] [ { FUNCTION | PROCEDURE | PACKAGE | PACKAGE BODY |
    TRIGGER| VIEW | TYPE | TYPE BODY | DIMENSION | JAVA CLASS } [schema.]nome ]
LNO
PARAMETERS [nome_parametro]
PNO
REL[EASE]
REPF[OOTER]
REPH[EADER]
SGA
SPOO[L]
SQLCODE
TTI[TLE]
USER
```

**DESCRIZIONE** SHOW visualizza il valore di una (o di tutte) caratteristica di SET o degli altri oggetti SQL\*PLUS. Più di un oggetto (quindi, più di una caratteristica di SET) può seguire la parola SHOW, e ciascuno viene visualizzato su una linea diversa. Le righe restituite vengono ordinate alfabeticamente.

*variabile\_sistema* può essere qualsiasi variabile impostata con il comando SET.

BTI[TLE] visualizza la definizione corrente di btitle.

ERR[ORS] mostra gli ultimi errori incontrati durante la compilazione di un oggetto memorizzato.

LNO mostra il numero di linea corrente (la linea della pagina attualmente visualizzata).

PARAMETERS mostra l'impostazione di un parametro specifico, o di tutti se non si specifica un parametro in particolare.

PNO visualizza il numero della pagina corrente.

REL[EASE] fornisce il numero della release di questa versione di Oracle.

REPF[OOTER] mostra il piè di pagina del report.

REPH[EADER] mostra l'intestazione del report.

SGA mostra le allocazioni di memoria correnti per la SGA.

SPOO[L] mostra la situazione di reindirizzamento dell'output su file (se è attivo o no). Vedere SPOOL.

SQLCODE mostra il numero del messaggio dell'errore Oracle più recente.

TTI[TLE] visualizza la definizione corrente di ttitle.

USER mostra l'ID dell'utente.

## **SHOWMODE (SQL\*PLUS)**

Vedere SET.

## **SIGN**

**VEDERE ANCHE** +, PRECEDENZE, Capitolo 8.

### **SINTASSI**

SIGN(*valore*)

**DESCRIZIONE** Vale 1 se *valore* è positivo, -1 se è negativo, 0 se vale zero.

### **ESEMPIO**

```
SIGN(33) = 1  
SIGN(-.6) = -1  
SIGN(0) = 0
```

## **SIN**

**VEDERE ANCHE** ACOS; ASIN; ATAN; ATAN2; COS; COSH; NUMERI, FUNZIONI; SINH; TAN; Capitolo 8.

### **SINTASSI**

SIN(*valore*)

**DESCRIZIONE** SIN restituisce il seno del *valore* di un angolo espresso in radianti.

**ESEMPIO**

```
select SIN(30*3.141593/180) Seno -- seno di 30 gradi in radianti from Dual;
```

**SINH**

**VEDERE ANCHE** ACOS; ASIN; ATAN; ATAN2; COS; NUMERI, FUNZIONI; SIN; TAN; Capitolo 8.

**SINTASSI**

SINH(*valore*)

**DESCRIZIONE** SINH restituisce il seno iperbolico del *valore* di un angolo.

**SINTASSI**

La sintassi è un insieme di regole che determinano le modalità di scrittura di un’istruzione valida per un particolare linguaggio di programmazione, come SQL.

**SMON**

Il System Monitor è uno dei processi in background di Oracle, utilizzato per eseguire il recupero e il cleanup dei segmenti temporanei inutilizzati. Vedere BACKGROUND, PROCESSO.

**SNAPSHOT**

“Snapshot” è il vecchio termine utilizzato per una vista materializzata. Vedere Capitolo 23, CREATE MATERIALIZED VIEW e CREATE MATERIALIZED VIEW LOG.

**SOSTITUZIONE**

Vedere &, &&, ACCEPT, DEFINE.

**SOUNDEX**

**VEDERE ANCHE** LIKE, Capitolo 7 e Capitolo 24.

**SINTASSI**

SOUNDEX(*stringa*)

**DESCRIZIONE** SOUNDEX trova tutte le parole che hanno un suono simile a una *stringa* di esempio. SOUNDEX fa alcune assunzioni sulla pronuncia comune di lettere e combinazioni di lettere. Le due parole da confrontare devono cominciare con la stessa lettera. Si possono eseguire ricerche di tipo SOUNDEX di singole parole all’interno di stringhe testuali negli indici CONTEXT. Vale solo per la lingua inglese.

## ESEMPIO

```
SOUNDEX('SEPP');
COGNOME          NOME           TELEFONO
-----
SZEP             FELICIA        214-522-8383
SEP              FELICIA        214-522-8383
```

## SPACE (SQL\*PLUS)

*Vedere SET.*

## SPOOL

**VEDERE ANCHE** SET, Capitolo 6.

### SINTASSI

```
SPO[OL] [file|OFF|OUT];
```

**DESCRIZIONE** SPOOL avvia o interrompe lo spooling (copia) dell'output di SQL\*PLUS su un file host di sistema o sulla stampante di sistema. SPOOL file reindirizza l'output di SQL\*PLUS al file indicato. Qualora non venga specificato il tipo di file, SPOOL ne aggiunge uno predefinito in maniera analoga a quanto fa SAVE. Solitamente si tratta di .LST ma con alcune variazioni di host. OFF interrompe lo spooling. OUT interrompe lo spooling e invia il file alla stampante. Per reindirizzare l'output a un file senza visualizzarlo, si imposti SET TERMOUT OFF nel file di avvio che precede l'istruzione SQL, ma prima del comando SPOOL. SPOOL senza parametri mostra il nome del file di spool corrente (o del più recente).

## SQL

SQL (Structured Query Language) è il linguaggio ANSI standard utilizzato per manipolare le informazioni di un database relazionale, e impiegato dai sistemi di gestione dei database relazionali Oracle e IBM DB2. SQL si pronuncia "siquel", anche se è ormai accettata la pronuncia "S.Q.L.".

## SQL\*LOADER

SQL\*LOADER è un'utility impiegata per il caricamento dei dati in un database Oracle. *Vedere* SQLldr e il Capitolo 21.

## SQL\*PLUS

*Vedere* SQLplus.

## SQLCASE (SQL\*PLUS)

*Vedere* SET.

## SQLCODE

**VEDERE ANCHE** ECCEZIONI, SQLERRM.

### SINTASSI

*variabile* := SQLCODE

**DESCRIZIONE** SQLCODE è una funzione di errore che restituisce il numero dell'ultimo errore, ma al di fuori di una sezione di gestione delle eccezioni (*vedere* ECCEZIONI e WHEN) restituisce sempre 0. Questo dipende dal fatto che l'occorrenza di un'eccezione (ovvero, quando SQLCODE è diversa da zero) dovrebbe trasferire immediatamente il controllo alla sezione di gestione delle eccezioni del blocco PL/SQL. Una volta in questa sezione, è possibile verificare il valore di SQLCODE o utilizzarlo per assegnare un valore a una variabile.

SQLCODE non può essere usato in un'istruzione SQL (per esempio, per inserire il valore del codice di errore in una tabella di errori). Tuttavia, è possibile impostare una variabile a SQLCODE e utilizzarla quest'ultima in un'istruzione SQL:

```
sql_error := sqlcode;
insert into PROBLEMLOG values (sql_error);
```

## SQLCONTINUE (SQL\*PLUS)

*Vedere* SET.

## SQLERRM

**VEDERE ANCHE** ECCEZIONI, EXCEPTION\_INIT, PRAGMA, SQLCODE.

### SINTASSI

SQLERRM[(*intero*)]

**DESCRIZIONE** Se non si specifica *intero*, SQLERRM restituisce il messaggio d'errore relativo al SQLCODE corrente. Se invece si specifica un intero, viene restituito il messaggio d'errore relativo a quel valore intero. Come SQLCODE, anche questa funzione non può essere utilizzata direttamente in un'istruzione SQL, ma può essere utile in un assegnamento:

```
error_message := sqlerrm;
```

o per ottenere il messaggio d'errore associato al codice 1403:

```
error_message := sqlerrm(1403);
```

Al di fuori di una sezione di gestione delle eccezioni (*vedere* SQLCODE), questa funzione senza un argomento intero restituisce sempre "normal, successful completion". In una sezione di gestione delle eccezioni, si ottiene uno dei messaggi seguenti.

- Il messaggio associato a un errore di Oracle.
- Le parole "User-defined exception" per un'eccezione sollevata esplicitamente dall'utente quando non è stato assegnato un messaggio.
- Un messaggio definito dall'utente caricato con PRAGMA EXCEPTION\_INIT.

## SQLJ

SQLJ è un preprocessore che genera codice compatibile con JDBC. In generale, il codice SQLJ è più semplice da scrivere del codice JDBC, inoltre è più facile eseguirne il debug.

## SQLLDR

**VEDERE ANCHE** Capitolo 21.

### SINTASSI

Parametri per il comando SQLLDR:

Userid	Nome utente e password per il caricamento, separati da una barra.
Control	Nome del control file.
Log	Nome del file di log.
Bad	Nome del file non valido.
Discard	Nome del file di scarto.
Discardmax	Numero massimo di righe da scartare prima di interrompere il caricamento. L'impostazione predefinita consente tutte le eliminazioni.
Skip	Numero di righe logiche nel file di input da scartare prima di avviare il caricamento dei dati. Solitamente viene utilizzato durante i nuovi caricamenti dallo stesso file di input, che seguono un caricamento parziale. Il valore predefinito è 0.
Load	Numero di righe logiche da caricare. L'impostazione predefinita è tutte.
Errors	Numero di errori ammessi. Il valore predefinito è 50.
Rows	Numero di righe su cui eseguire contemporaneamente il commit. Questo parametro serve per suddividere la dimensione della transazione durante il caricamento. L'impostazione predefinita per i caricamenti a percorso convenzionale è 64, per i caricamenti a percorso diretto è tutte le righe.
Bindsizes	Dimensione dell'array di bind a percorso convenzionale espresso in byte. L'impostazione predefinita dipende dal sistema operativo.
Silent	Elimina i messaggi durante il caricamento.
Direct	Usa il caricamento a percorso diretto. L'impostazione predefinita è FALSE.
Parfile	Nome del file di parametri che contiene altre specifiche dei parametri di caricamento.
Parallel	Esegue il caricamento in parallelo. L'impostazione predefinita è FALSE.
File	File da cui allocare gli extent (per il caricamento in parallelo).
Skip_Unusable_Indexes	Consente di eseguire caricamenti in tabelle che hanno indici in stati inutilizzabili. L'impostazione predefinita è FALSE.
Skip_Index_Maintenance	Interrompe la gestione degli indici per i caricamenti a percorso diretto, lasciandoli in uno stato inutilizzabile. L'impostazione predefinita è FALSE.
Readsize	Dimensione del buffer di lettura espresso in byte; il valore predefinito è 1048576.

(segue)

*(continua)*

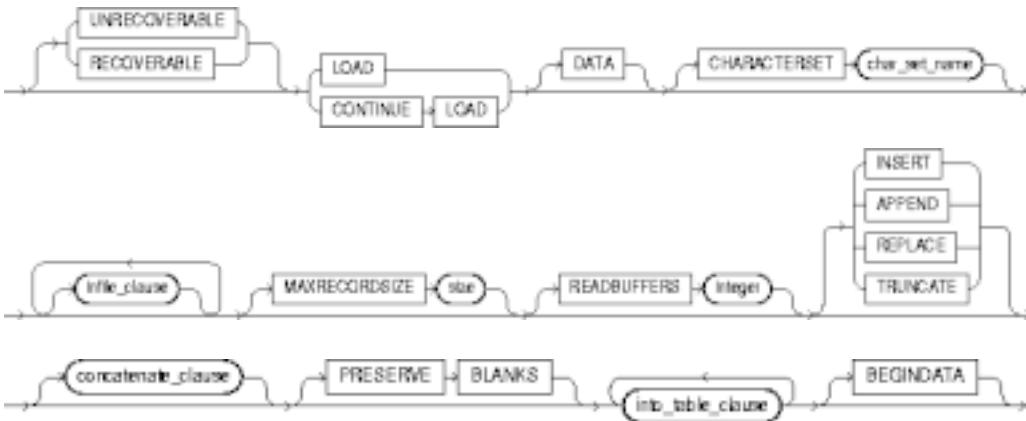
External_Table	Usa la tabella esterna per il caricamento. L'impostazione predefinita è NOT_USED, gli altri valori sono GENERATE_ONLY ed EXECUTE.
ColumnArrayRows	Numero di righe per array di colonna a percorso diretto. Il valore predefinito è 5000.
StreamSize	Dimensione del buffer di flusso a percorso diretto espressa in byte. Il valore predefinito è 256000.
Multithreading	Usa il multithreading nel percorso diretto
Resumable	Attiva o disattiva la possibilità del riutilizzo nella sessione corrente; l'impostazione predefinita è FALSE.
Resumable_Name	Stringa di testo impiegata per identificare le istruzioni riutilizzabili.
Resumable_Timeout	Tempo di attesa, in secondi per il parametro Resumable. Il valore predefinito è 7200.

**DESCRIZIONE** SQL\*LOADER carica i dati dai file esterni in tabelle del database Oracle. Per esempi, si *rimanda* al Capitolo 21. La sintassi del control file è:

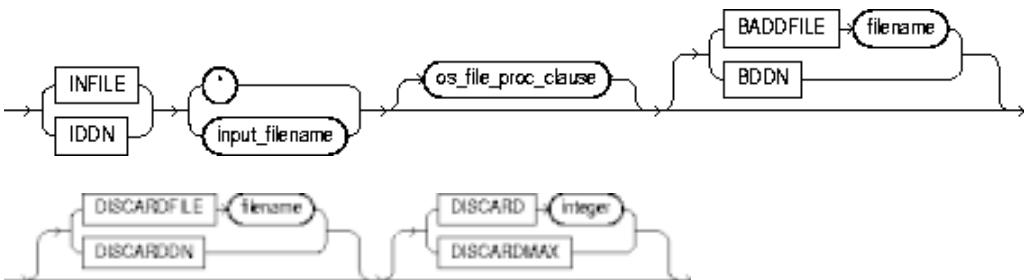
#### Options\_clause

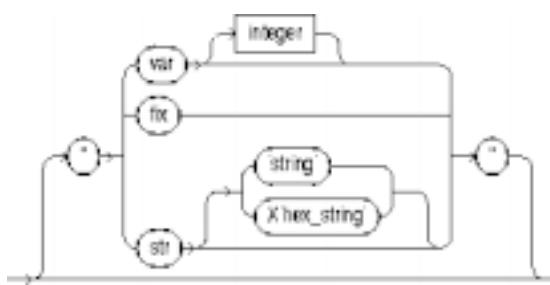
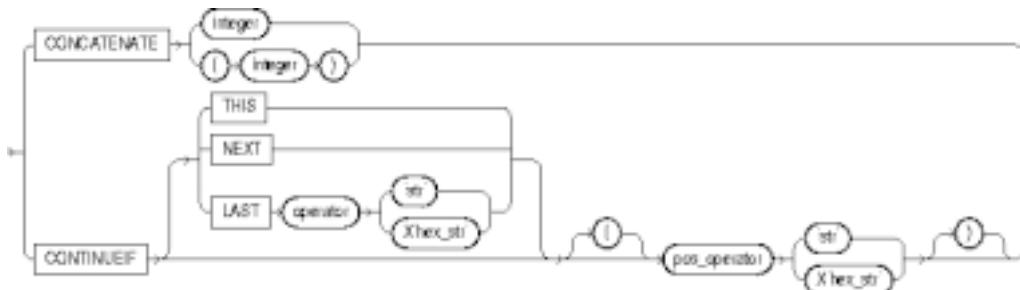
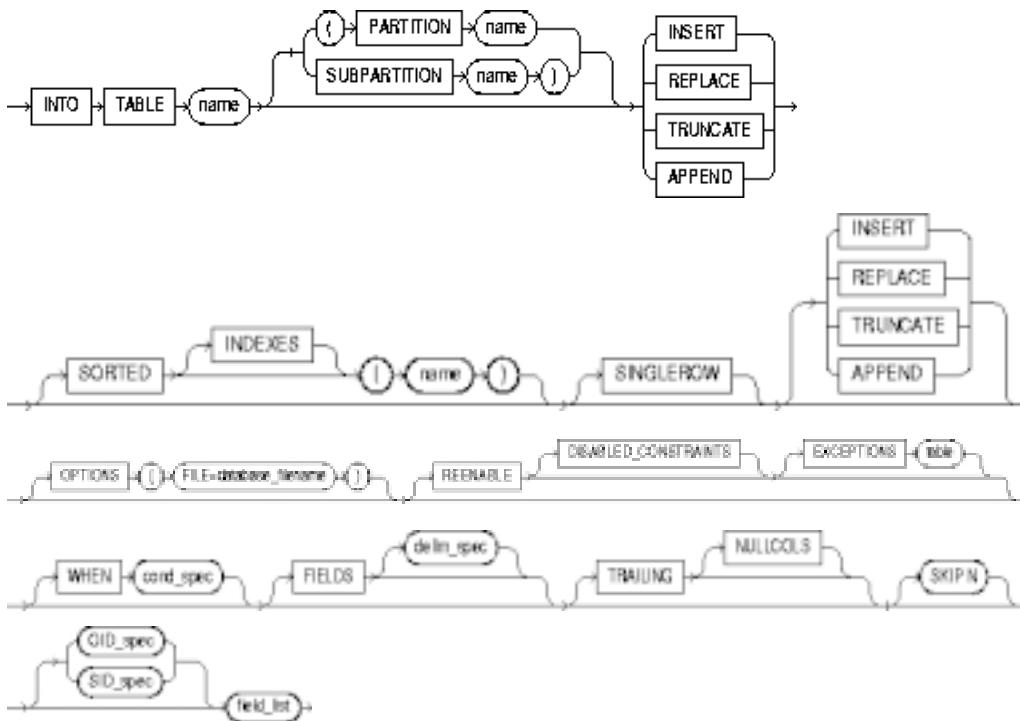


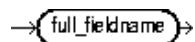
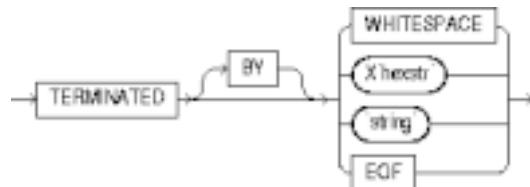
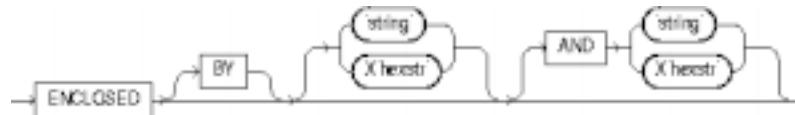
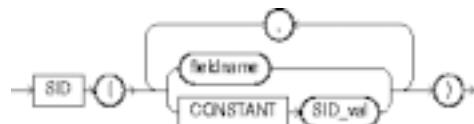
#### Load\_statement

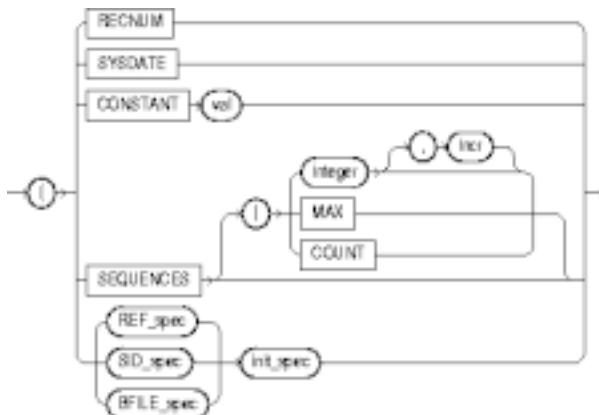
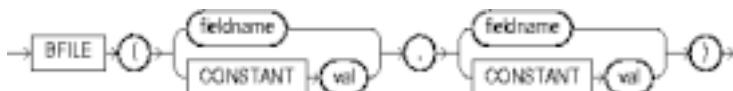


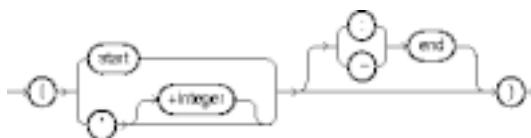
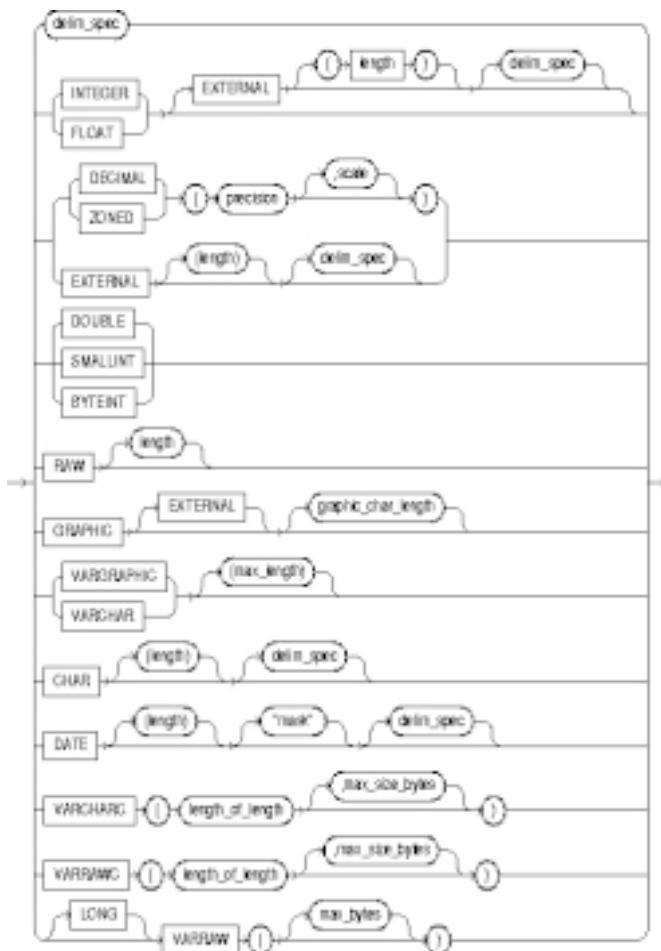
#### infile\_clause

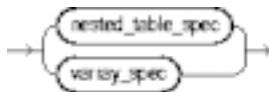
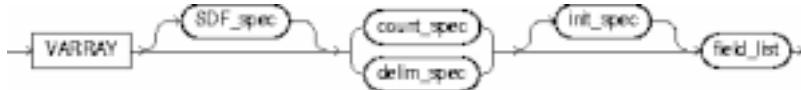
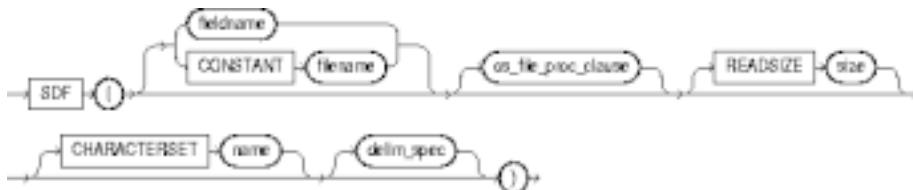
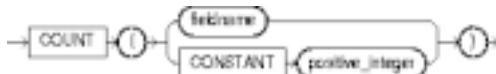


**on\_file\_proc\_clause****concatenate\_clause****into\_table\_clause**

**cond\_spec****delim\_spec****full\_fieldname****termination\_spec****enclosure\_spec****OID\_spec****SID\_spec**

**field\_list****d\_gen\_fid\_spec::=****REF\_spec****init\_spec****BFILE\_spec****filler\_fid\_spec**

**scalar fld spec****LOBFILE\_spec****pos\_spec****datatype\_spec**

**col\_obj\_fld\_spec****collection\_fld\_spec****nested\_table\_spec****VARRAY\_spec****SDF\_spec****count\_spec****SQLNUMBER (SQL\*PLUS)**

*Vedere SET.*

**SQLPLUS**

**VEDERE ANCHE** Capitoli 6, 14.

**SINTASSI**

```
SQLPLUS [utente[/password][@database] [@file] ] [-SILENT] |
  [/NOLOG] [-SILENT] | [-?]
```

**DESCRIZIONE** SQLPLUS avvia SQL\*PLUS. Se si inseriscono sia il nome utente sia la password, SQL\*PLUS si collega al database predefinito. Se si specifica solo il proprio nome utente, SQL\*PLUS richiede anche la password, che non viene mai visualizzata mentre la si inserisce. Se si specifica `@database` si ottiene la connessione al database specificato, invece che a quello predefinito. `@database` può essere ovunque, purché il computer cui si è collegati sia connesso a questo database tramite Oracle Net. Non devono esserci spazi tra `password` e `@database`. Per ulteriori informazioni su Oracle Net, si rimanda al Capitolo 22. `@file` eseguirà il file di avvio non appena SQL\*PLUS è stato caricato. Prima di `@file` deve esserci uno spazio. Se il nome utente e la password non vengono digitati sulla linea di comando con SQLPLUS, devono trovarsi sulla prima linea del file. Se sono presenti nel file, e li si inserisce nella linea di comando si ottiene un messaggio di errore che non influirà sulla corretta esecuzione del file di avvio. Per inserire il nome utente e la password all'inizio del file, li si separi tramite una barra obliqua (/):

GEORGE/MISTY

`/NLOG` avvia SQL\*PLUS, ma non connette l'utente al database Oracle. Per collegarsi a Oracle, si dovrà ricorrere a `CONNECT`. (*Vedere CONNECT*).

`-SILENT` elimina tutte le visualizzazioni delle schermate di SQL\*PLUS, compresi i prompt dei comandi e persino le informazioni sul collegamento a SQL\*PLUS e sul copyright. In questo modo, è possibile che altri programmi usino SQL\*PLUS in maniera trasparente.

`-?` visualizza la versione corrente e il numero del livello di SQL\*PLUS senza avviarlo.

### ESEMPIO

Ecco un avvio classico di SQL\*PLUS:

```
sqlplus george/misty
```

Per collegarsi contestualmente al database EDMESTON si utilizzi:

```
sqlplus george/misty@EDMESTON
```

Per avviare il file di report REPORT6 che contiene nome utente e password sulla prima linea si utilizzi:

```
sqlplus @report6
```

Per avviare il file di report REPORT6 che include il nome utente e la password nella prima linea sul database EDMESTON si utilizzi:

```
sqlplus george/misty@EDMESTON @report6
```

## SQLPREFIX (SQL\*PLUS)

*Vedere SET.*

## SQLPROMPT (SQL\*PLUS)

*Vedere SET.*

## SQLTERMINATOR (SQL\*PLUS)

Vedere SET.

## SQRT

**VEDERE ANCHE** POWER.

### SINTASSI

SQRT(*valore*)

**DESCRIZIONE** SQRT calcola la radice quadrata di valore.

### ESEMPI

SQRT(64) = 8

La radice quadrata di numeri negativi non è disponibile in Oracle.

## START

**VEDERE ANCHE** @, @@, ACCEPT, DEFINE, SPOOL, Capitolo 6.

### SINTASSI

STA[RT] *file* [*parametro*] [*parametro*]...

**DESCRIZIONE** START esegue il contenuto del file di avvio specificato (che prende questo nome poiché è avviato proprio da questo comando). Il file può contenere qualsiasi comando di SQL\*PLUS. Se non si specifica alcun tipo di file, START suppone si tratti di .sql. Gli eventuali parametri che seguono il nome del file vengono sostituiti in variabili contenute nel file di avvio; le variabili devono avere come nomi &1, &2, &3 e così via e ricevono i parametri in ordine da sinistra a destra. Ogni occorrenza di &1 nel file di avvio ottiene il primo parametro. I parametri costituiti da più di una parola possono essere racchiusi tra apici (""); altrimenti, saranno limitati a una singola parola o a un singolo numero.

### ESEMPIO

```
start checkout INNUMERACY
```

in cui il file checkout.sql contiene una variabile di nome &1, in cui verrà sostituita la stringa 'INNUMERACY'.

## STDDEV

**VEDERE ANCHE** FUNZIONI DI GRUPPO, VARIANCE, Capitolo 8.

### SINTASSI

STDDEV ( [ DISTINCT | ALL ] *valore* ) [OVER ( *clausola\_analitica* )]

**DESCRIZIONE** STDDEV fornisce la deviazione standard dalla norma di valori in un gruppo di righe. Nel calcolo vengono ignorati eventuali valori NULL.

## STDDEV\_POP

**VEDERE ANCHE** FUNZIONI DI GRUPPO, STTDEV, STDDEV\_SAMP.

### SINTASSI

STDDEV\_POP ( *espr* ) [OVER ( *clausola\_analitica* )]

**DESCRIZIONE** STDDEV\_POP calcola la deviazione standard della popolazione e restituisce la radice quadrata della varianza della popolazione.

## STDDEV\_SAMP

**VEDERE ANCHE** FUNZIONI DI GRUPPO, STTDEV, STDDEV\_POP.

### SINTASSI

STDDEV\_POP ( *espr* ) [OVER ( *clausola\_analitica* )]

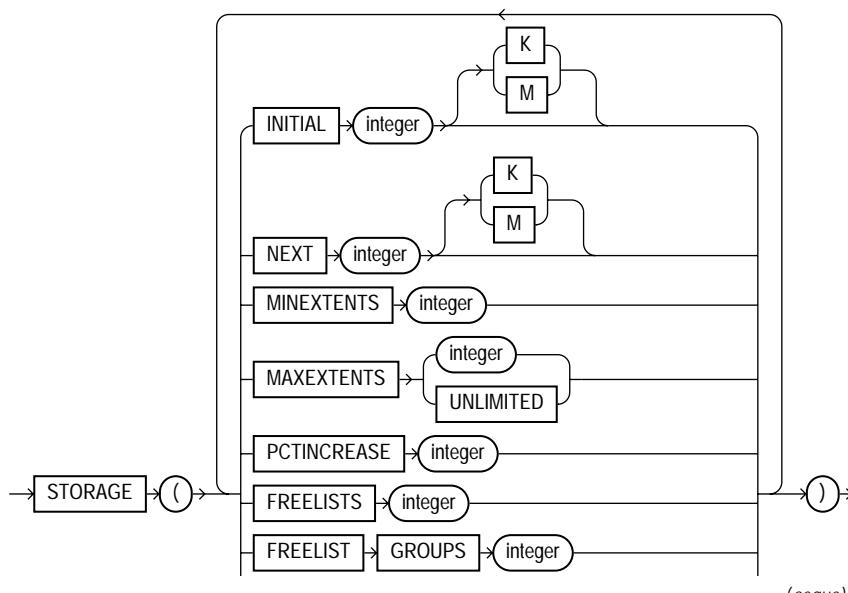
**DESCRIZIONE** STDDEV\_SAMP calcola la deviazione standard del campione cumulativo e restituisce la radice quadrata della varianza del campione.

## STORAGE

**VEDERE ANCHE** BLOCK, CREATE CLUSTER, CREATE INDEX, CREATE ROLLBACK SEGMENT, CREATE MATERIALIZED VIEW, CREATE MATERIALIZED VIEW LOG, CREATE TABLE, CREATE TABLESPACE e le analoghe istruzioni ALTER.

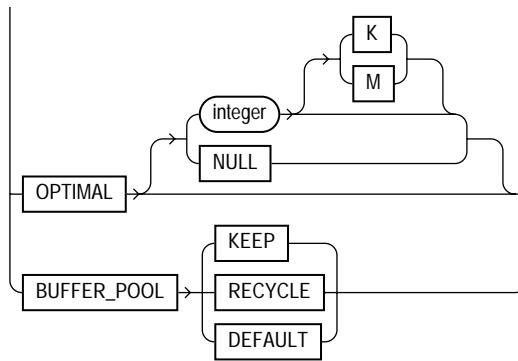
### SINTASSI

storage\_clause



(segue)

(continua)



**DESCRIZIONE** La clausola **STORAGE** può essere utilizzata in qualsiasi delle istruzioni **CREATE** e **ALTER** riportate nella sezione “Vedere anche”. Non essendo un’istruzione SQL, non può rimanere da sola. Nei paragrafi seguenti, un blocco sarà un blocco di database (*vedere BLOCK*) con una dimensione che varia in base al sistema operativo in uso e ai parametri di inizializzazione del database.

**INITIAL** alloca il primo extent di spazio in un oggetto. Se non si specifica questa opzione, per default si usano cinque blocchi di dati. L’extent iniziale più piccolo che si possa allocare è di due blocchi di dati, mentre il più grande dipende dal sistema operativo. È possibile esprimere queste dimensioni sotto forma di numeri interi semplici o di interi seguiti da **K** o **M** per indicare rispettivamente kilobyte o megabyte.

**NEXT** rappresenta la dimensione dell’extent allocato una volta riempito l’extent iniziale. Se non viene specificato altro, l’impostazione predefinita è di cinque blocchi di dati. L’extent successivo più piccolo che si possa allocare è di un blocco di dati, mentre il più grande dipende dal sistema operativo. Si possono utilizzare **K** e **M** per i kilobyte e i megabyte.

**PCTINCREASE** controlla il tasso di crescita degli extent che seguono il secondo. Se viene impostato a 0, ogni extent aggiuntivo avrà la stessa dimensione del secondo, specificato da **NEXT**. Se **PCTINCREASE** è un numero intero positivo, ogni extent successivo sarà più grande di quello precedente della percentuale specificata. Per esempio, se **PCTINCREASE** vale 50 (l’impostazione predefinita), ogni extent aggiuntivo sarà del 50 per cento più grande del precedente. **PCTINCREASE** non può essere negativo. Il valore minimo è zero mentre il massimo dipende dal sistema operativo. Oracle arrotonda la dimensione degli extent al multiplo intero successivo della dimensione di blocco impostata dal sistema operativo.

Per default, **MINEXTENTS** vale 1 (o 2 per i segmenti di rollback) se non viene specificato, e ciò significa che quando si crea l’oggetto, viene allocato solo l’extent iniziale. Un numero maggiore di 1 crea diversi extent (che non devono necessariamente essere attigui su disco, come fa ogni extent) e la dimensione di ogni extent viene determinata dai valori impostati con **INITIAL**, **NEXT** e **PCTINCREASE**. Tutti questi extent verranno allocati quando si crea l’oggetto.

**MAXEXTENTS** impone il numero massimo di extent allocabili. Il numero minimo è 1, il massimo dipende dal sistema operativo. Si può specificare un valore **MAXEXTENTS** come **UNLIMITED**.

**OPTIMAL** impone la dimensione ottimale, espressa in byte, di un segmento di rollback. Si possono utilizzare **K** e **M** per i kilobyte e i megabyte. Oracle annulla dinamicamente l’allocazione degli extent nel segmento di rollback per mantenere la dimensione ottimale. **NULL** significa che Oracle non annulla mai l’allocazione degli extent nei segmenti di rollback e questo è il compor-

tamento predefinito. Si deve fornire una dimensione maggiore o uguale allo spazio iniziale allocato per il segmento di rollback dai parametri MINEXTENTS, INITIAL, NEXT e PCTINCREASE.

FREELIST GROUPS fornisce il numero di gruppi di free list, con un valore predefinito pari a 1. Questa impostazione si applica all'opzione Real Application Clusters di Oracle per le tabelle, i cluster e gli indici.

FREELISTS imposta il numero di free list in ogni gruppo.

BUFFER\_POOL imposta il buffer pool in cui verranno letti i blocchi della tabella. Per default, tutti i blocchi vengono letti nel pool DEFAULT. Si possono creare pool separati per i blocchi che si desidera mantenere in memoria più a lungo (il pool di conservazione) oppure per quelli che si desidera rimuovere rapidamente dalla memoria (il pool di riciclo).

## STORE

**VEDERE ANCHE** SAVE, START, Capitolo 14.

### SINTASSI

```
STORE SET file[.est] [ CREATE ] | REPLACE | APPEND ]
```

**DESCRIZIONE** STORE salva tutte le impostazioni ambientali di SQLPLUS in un file.

### ESEMPIO

Il comando seguente salva le impostazioni ambientali di SQLPLUS correnti in un file di nome SETTINGS.SQL.

```
store settings.sql
```

## STRUCTURED QUERY LANGUAGE

*Vedere* SQL.

## SUBQUERY

Una query (ovvero, un'istruzione select) può essere utilizzata all'interno di altre istruzioni SQL (dette istruzioni padre o esterne), tra cui CREATE TABLE, delete, insert, select, update e la versione SQL\*PLUS del comando COPY, per definire le righe e le colonne che il padre utilizzerà nella propria esecuzione. I risultati della query figlia (definita anche subquery) non vengono visualizzati ma vengono passati all'istruzione SQL esterna. Questo costrutto deve rispettare le regole seguenti.

- Nei comandi update e CREATE TABLE, la subquery deve restituire un valore per ogni colonna da inserire o aggiornare. I valori vengono quindi utilizzati dall'istruzione SQL esterna per inserire o aggiornare le righe.
- Una subquery non può contenere le clausole order by e for update of.
- Nella clausola where di un'istruzione select viene utilizzata una subquery "correlata" che si riferisce all'alias della tabella utilizzata dal comando di selezione esterno. Questa subquery viene verificata una volta per ogni riga valutata per la selezione dall'istruzione select esterna (una subquery standard valutata una sola volta per la query padre). Per ulteriori dettagli, si *rimanda* al Capitolo 12.

A parte le restrizioni appena citate, una subquery obbedisce alle stesse leggi che governano le normali istruzioni select.

## SUBSTR

**VEDERE ANCHE** ||; CARATTERI, FUNZIONI; INSTR; Capitolo 7.

### SINTASSI

SUBSTR(*stringa*, *inizio* [*.conta*])

**DESCRIZIONE** SUBSTR taglia una parte di una stringa partendo dal carattere la cui posizione è *inizio* e terminando dopo un numero di caratteri definito da *conta*. Se non si specifica quest'ultimo parametro, la stringa viene tagliata a partire da *inizio* fino alla fine.

### ESEMPIO

SUBSTR('RIVISTA',5)

produce:

STA

## SUFFIX (SQL\*PLUS)

Vedere SET.

## SUGGERIMENTI

All'interno di una query, è possibile specificare dei suggerimenti che dirigono l'ottimizzatore basato sui costi durante l'elaborazione della query stessa. Per specificare un suggerimento, si utilizza la sintassi seguente. Si digiti la stringa che segue subito dopo la parola chiave select:

/\*+

Quindi si aggiunga un suggerimento, per esempio:

FULL(biblioteca)

Si termini il suggerimento con la stringa seguente:

\*/

Per una descrizione di tutti i suggerimenti disponibili e dell'impatto che esercitano sull'elaborazione delle query, si rimanda al Capitolo 38.

## SUM

**VEDERE ANCHE** COMPUTE, FUNZIONI DI GRUPPO, Capitolo 8.

### SINTASSI

SUM([DISTINCT | ALL] *valore*) [OVER (*clausola\_analitica*)]

**DESCRIZIONE** SUM è la somma di tutti i *valori* di un gruppo di righe. DISTINCT fa sì che SUM sommi ogni valore unico al totale una sola volta; opzione generalmente poco utile.

## SYNONYM

Un sinonimo è un nome assegnato a una tabella, a una vista o a una sequenza per semplificare i successivi riferimenti a quella stessa risorsa. Se si ha la possibilità di accedere a una tabella di un altro utente, è possibile crearne un sinonimo e impiegare quest'ultimo per riferirsi alla tabella senza introdurre il nome dell'altro utente come qualificatore. Vedere i Capitoli 22 e 37.

## SYS (UTENTE ORACLE)

SYS è uno degli utenti DBA creati all'atto dell'installazione e dell'inizializzazione del sistema di database (l'altro è SYSTEM). SYS possiede la maggior parte delle tabelle del dizionario dati, mentre SYSTEM è il proprietario delle viste create su queste tabelle di base.

## SYS\_CONNECT\_BY\_PATH

**VEDERE ANCHE** CONNECT BY, Capitolo 13.

### SINTASSI

SYS\_CONNECT\_BY\_PATH (*colonna*, *car*)

**DESCRIZIONE** SYS\_CONNECT\_BY\_PATH è valida soltanto nelle query gerarchiche. Restituisce il percorso dalla radice al nodo di un valore di colonna, con i valori di colonna separati da *car* per ogni riga restituita dalla condizione CONNECT BY.

## SYS\_CONTEXT

**VEDERE ANCHE** CREATE CONTEXT.

### SINTASSI

SYS\_CONTEXT ( *spazio\_nomi* , *parametro* [ , *lunghezza* ] )

**DESCRIZIONE** SYS\_CONTEXT restituisce il valore di *parametro* associato allo *spazio\_nomi* contestuale.

## SYS\_DBURIGEN

**VEDERE ANCHE** Capitolo 41.

### SINTASSI

```
SYS_DBURIGEN
( { colonna | attributo } [rowid]
  [. { colonna | attributo } [rowid]]...
  [. 'text ( )'] )
```

**DESCRIZIONE** SYS\_DBURIGEN accetta una o più colonne oppure uno o più attributi come propri argomenti, e facoltativamente un RowID, generando l'URL di un tipo di dati DBUriType per una colonna o un oggetto riga particolari. Quindi, si potrà utilizzare questo URL per recuperare un documento XML dal database.

Tutte le colonne o gli attributi indicati devono risiedere nella stessa tabella. Inoltre, devono eseguire la funzione di una chiave primaria. In altre parole, non devono effettivamente corrispondere alle chiavi primarie della tabella, ma devono fare riferimento a un valore unico. Se si specificano più colonne, tutte tranne quella finale identificano le righe del database mentre l'ultima colonna specificata identifica la colonna all'interno delle righe.

Per default, l'URL punta a un documento XML formattato. Se si desidera che l'URL punti solo al testo del documento, si specifichi l'opzione facoltativa 'text()'. (In questo contesto XML, la parola "text" scritta in minuscolo è una parola chiave e non un segnaposto sintattico).

## SYS\_EXTRACT\_UTC

**VEDERE ANCHE** Capitolo 8.

### SINTASSI

SYS\_EXTRACT\_UTC ( *data/ora\_con\_fuso\_orario* )

**DESCRIZIONE** SYS\_EXTRACT\_UTC estrae il valore UTC (Coordinated Universal Time) da una data/ora con la differenza di fuso orario.

## SYS\_GUID

### SINTASSI

SYS\_GUID ( )

**DESCRIZIONE** SYS\_GUID genera e restituisce un identificativo unico (un valore di tipo RAW) costituito da 16 byte. Sulla maggior parte delle piattaforme, l'identificativo creato è costituito da un identificatore host e da un processo o un thread del processo oppure da un thread che invoca la funzione, e da un valore non ripetuto (sequenza di byte) per quel processo o thread.

## SYS\_TYPEID

**VEDERE ANCHE** Capitolo 33.

### SINTASSI

SYS\_TYPEID ( *valore\_tipo Oggetto* )

**DESCRIZIONE** SYS\_TYPEID restituisce l'ID del tipo più specifico dell'operando. Questo valore viene utilizzato principalmente per identificare la colonna discriminante del tipo che funge da base per una colonna sostituibile. Per esempio, si può utilizzare il valore restituito da SYS\_TYPEID per costruire un indice sulla colonna discriminante del tipo.

## SYS\_XMLAGG

**VEDERE ANCHE** SYS\_XMLGEN, Capitolo 41.

### SINTASSI

SYS\_XMLAGG ( *espr* [ *fmt* ] )

**DESCRIZIONE** SYS\_XMLAGG aggrega tutti i documenti o i frammenti XML rappresentati da *espr* e produce un unico documento XML. Aggiunge un nuovo elemento di inclusione con un nome predefinito come ROWSET. Se si desidera formattare il documento XML in modo differente, si specifichi *fmt*, che è un'istanza dell'oggetto SYS.XMLGenFormatType.

## SYS\_XMLGEN

**VEDERE ANCHE** SYS\_XMLAGG, Capitolo 41.

### SINTASSI

SYS\_XMLGEN ( *espr* [ *fmt* ] )

**DESCRIZIONE** SYS\_XMLGEN accetta un'espressione che viene valutata a una particolare riga e colonna del database, restituendo un'istanza del tipo SYS.XMLType che contiene un documento XML. *espr* può essere un valore scalare, un tipo definito dall'utente oppure un'istanza XMLType.

Se *espr* è un valore scalare, la funzione restituisce un elemento XML che contiene il valore scalare. Se *espr* è un tipo, la funzione associa gli attributi del tipo definito dall'utente agli elementi XML. Se *espr* è un'istanza XMLType, la funzione include il documento in un elemento XML; il nome di tag predefinito di questo elemento è ROW.

Per default, gli elementi del documento XML corrispondono agli elementi di *espr*. Per esempio, se *espr* si risolve in un nome di colonna, l'elemento XML di inclusione avrebbe lo stesso nome di colonna. Se si desidera formattare il documento XML in modo differente, si specifichi *fmt*, che è un'istanza dell'oggetto SYS.XMLGenFormatType.

## SYSDATE

**VEDERE ANCHE** Capitolo 9.

### SINTASSI

SYSDATE

**DESCRIZIONE** SYSDATE restituisce la data e l'ora correnti.

### ESEMPIO

```
select SYSDATE from DUAL;
```

## SYSTEM (TABLESPACE ORACLE)

SYSTEM è il nome assegnato alla prima tablespace di un database. Essa contiene il dizionario dati.

## SYSTEM (UTENTE ORACLE)

SYSTEM è uno degli utenti DBA creati all'atto dell'installazione e dell'inizializzazione del sistema di database (l'altro è SYS). SYS possiede la maggior parte delle tabelle del dizionario dati, mentre SYSTEM è il proprietario delle viste create su queste tabelle di base.

## SYSTEM GLOBAL AREA (SGA)

SGA è un'area di memoria condivisa che rappresenta il centro dell'attività di Oracle mentre il database è in esecuzione. La dimensione della SGA (e le prestazioni del sistema) dipendono dai valori dei parametri di inizializzazione delle variabili. La SGA stabilisce una comunicazione tra l'utente e i processi in background.

## SYSTIMESTAMP

**VEDERE ANCHE** Capitolo 9.

### SINTASSI

SYSTIMESTAMP

**DESCRIZIONE** SYSTIMESTAMP restituisce la data e l'ora correnti.

### ESEMPIO

```
select SYSTIMESTAMP from DUAL;
```

## TABELLA

Una tabella è la struttura di memorizzazione dei dati fondamentali per i sistemi di gestione dei database relazionali. Una tabella è costituita da una o più unità d'informazione (righe), ciascuna contenente gli stessi tipi di valori (colonne). Vedere CREATE TABLE.

## TABELLA ANNIDATA

Una tabella annidata è, come suggerisce il nome stesso, una tabella contenuta all'interno di un'altra. In questo caso, si tratta di una tabella rappresentata sotto forma di una colonna all'interno di un'altra tabella. È possibile avere diverse righe della tabella annidata associate a ciascuna riga della tabella principale. Non c'è alcun limite al numero di voci per riga. Per dettagli relativi alla creazione, uso e query di tabelle annidate, si consulti il Capitolo 31.

## TABELLA DI SOLO INDICE

Una *tabella di solo indice* mantiene i propri dati ordinati secondo i valori della colonna di chiave primaria della tabella. Una tabella di solo indice consente di memorizzare i dati dell'intera tabella in un indice. Un indice normale memorizza nell'indice solo le colonne indicizzate; una tabella di solo indice memorizza nell'indice tutte le colonne della tabella.

Per creare una tabella di solo indice, è necessario utilizzare la clausola organization index del comando create table, come mostrato nell'esempio seguente:

---

```
create table STRANO (
Citta          VARCHAR2(13),
DataCampione   DATE,
Mezzogiorno    NUMBER(4,1),
Mezzanotte     NUMBER(4,1),
Precipitazione NUMBER,
constraint STRANO_PK PRIMARY KEY (Citta, DataCampione))
organization index;
```

Per creare STRANO come tabella di solo indice, occorre creare su di essa un vincolo PRIMARY KEY.

In generale, le tabelle di solo indice sono adatte per le tabelle associative in relazioni di tipo molti a molti. Idealmente, una tabella di solo indice avrà poche colonne (o nessuna) che non fanno parte della sua chiave primaria.

*Vedere* il Capitolo 18 e CREATE TABLE.

## TABELLA ESTERNA

Una tabella esterna è un file che è esterno al database cui si può accedere tramite i comandi SQL. Per i dettagli sulla creazione e l'uso delle tabelle esterne, si rimanda al Capitolo 25.

## TABELLA OGGETTO

Una tabella oggetto è una tabella in cui ogni riga è un oggetto riga. *Vedere* il Capitolo 33 e la parte relativa al comando CREATE TABLE.

## TABELLA PARTIZIONATA

Una tabella partizionata è una tabella le cui righe siano state partizionate su diverse tabelle più piccole.

## TABLE, PL/SQL

**VEDERE ANCHE** DATI, TIPI; RECORD (PL/SQL).

### SINTASSI

```
TYPE nuovo_tipo IS TABLE OF
{tipo | colonna.tabella%TYPE}[NOT NULL]
INDEX BY BINARY_INTEGER;
```

**DESCRIZIONE** Una dichiarazione TABLE crea un nuovo tipo che potrà essere utilizzato nella dichiarazione di variabili di quel tipo. Una tabella PL/SQL ha un'unica colonna e una chiave intera e può avere un numero qualsiasi di righe. Il tipo di dati della colonna può essere un tipo standard PL/SQL (compresi i RECORD ma escluse altre tabelle), o può essere un riferimento a un tipo di una colonna particolare già utilizzato in un'altra tabella del database. I campi possono anche avere un qualificatore NOT NULL, che specifica che il campo deve sempre avere un valore definito.

La clausola index by binary integer è indispensabile e ricorda che l'indice è necessariamente intero. Si può fare riferimento a qualsiasi riga della tabella menzionando il suo indice tra parentesi.

Se si fa riferimento a un indice cui non è stato ancora assegnato alcun dato, PL/SQL solleva l'eccezione NO\_DATA\_FOUND.

## TABLESPACE

Una tablespace è un file o un insieme di file utilizzato per memorizzare dati di Oracle. Un database Oracle è costituito dalla tablespace SYSTEM e molto probabilmente da altre tablespace. Vedere i Capitoli 20 e 40.

## TABLESPACE GESTITE A LIVELLO LOCALE

Una tablespace gestita a livello locale mantiene le proprie informazioni sull'uso degli extent in bitmap all'interno dei file di dati della tablespace. Per default, le tablespace memorizzano le proprie informazioni sull'uso degli extent nel dizionario dati (e sono indicate come "gestite a livello di dizionario"). Vedere CREATE TABLESPACE.

## TABLESPACE TRASPORTABILE

Una tablespace trasportabile è una tablespace che può essere "scollegata" da un database e "collegata" in un altro. Per essere trasportabile, una tablespace, o un insieme di tablespace, deve essere autonoma. Il gruppo di tablespace non può contenere oggetti che fanno riferimento a oggetti contenuti in altre tablespace. Pertanto, se si trasporta una tablespace contenente indici, si dovrà spostare la tablespace che contiene le tabelle base degli indici come parte del medesimo insieme di tablespace trasportabili. Migliore è l'organizzazione e la distribuzione degli oggetti tra le tablespace, più semplice sarà la creazione di un gruppo autonomo di tablespace da trasportare.

Per trasportare le tablespace, è necessario creare un insieme di tablespace, copiarlo o spostarlo nel nuovo database quindi collegare il gruppo al nuovo database. I database devono risiedere sul medesimo sistema operativo, con la stessa versione di Oracle e del set di caratteri.

### Generazione di un gruppo di tablespace trasportabili

Un gruppo di tablespace trasportabili contiene tutti i file di dati delle tablespace da spostare insieme a file di esportazione per i metadati di queste tablespace. Facoltativamente, è possibile scegliere di includere vincoli di integrità referenziale nel gruppo di tablespace trasportabili. Se si decide di utilizzare vincoli di integrità referenziale, l'insieme di tablespace trasportabili si allargherà per comprendere le tabelle necessarie a gestire le relazioni di chiave. L'integrità referenziale è facoltativa perché si potrebbero avere le stesse tabelle di codici in più database. Per esempio, si può programmare di spostare una tablespace dal proprio database di prova al database di produzione. Se nel database di prova è presente una tabella di nome PAESE, si potrebbe avere una tabella PAESE identica nel database di produzione. Dato che le tabelle dei codici sono identiche nei due database, non sarà necessario trasportare questa parte dei vincoli di integrità referenziale. Si potrebbe trasportare la tablespace, quindi attivare nuovamente l'integrità referenziale nel database di destinazione una volta spostata la tablespace, semplificando così la creazione dell'insieme di tablespace trasportabili.

Per stabilire se un gruppo di tablespace è autonomo, si esegua la procedura TRANSPORT\_SET\_CHECK del package DBMS\_TTS. Questa procedura accetta due parametri di input: l'insieme di tablespace e un flag booleano impostato a TRUE, se si desidera che i vincoli di integrità referenziale vengano considerati. Nell'esempio seguente, i vincoli di integrità referenziale non

vengono presi in considerazione per la combinazione delle tablespace AGG\_DATA e AGG\_INDEXES:

```
execute DBMS_TTS.TRANSPORT_SET_CHECK('AGG_DATA,AGG_INDEXES','FALSE');
```

Se nell'insieme specificato si verificano violazioni di autonomia, Oracle popolerà la vista del dizionario dati TRANSPORT\_SET\_VIOLATIONS. Se al contrario non vi sono violazioni, questa vista rimarrà vuota.

Dopo aver selezionato un gruppo autonomo di tablespace, queste dovranno essere trasformate in tablespace di sola lettura, come mostrato di seguito:

```
alter tablespace AGG_DATA read only;
alter tablespace AGG_INDEXES read only;
```

Quindi, si dovranno esportare i metadati delle tablespace, con i parametri di Export TRANSPORT\_TABLESPACES e TABLESPACES:

```
exp TRANSPORT_TABLESPACE=Y TABLESPACES=(AGG_DATA,AGG_INDEXES) CONSTRAINTS=N GRANTS=Y
TRIGGERS=N
```

Come mostrato nell'esempio, è possibile specificare se i trigger, i vincoli e le concessioni verranno esportati insieme ai metadati delle tablespace. Inoltre, si dovrebbero notare i nomi degli account che possiedono oggetti nel gruppo di tablespace trasportabili. Ora si possono copiare i file di dati delle tablespace in un'area separata. Se necessario, è possibile riportare le tablespace nella modalità lettura-scrittura nel database corrente. Dopo aver creato il gruppo di tablespace trasportabili, si potranno spostare i suoi file (incluso quello di esportazione) in un'area cui il database di destinazione può accedere.

#### **Collegamento al gruppo di tablespace trasportabili**

Dopo che il gruppo di tablespace trasportabili è stato spostato in un'area accessibile al database di destinazione, si potrà collegare il gruppo al database di destinazione. Per prima cosa, occorre utilizzare Import per importare i metadati esportati:

```
imp TRANSPORT_TABLESPACE=Y DATAFILES=(agg_data.dbf, agg_indexes.dbf)
```

Nell'importazione è necessario specificare i file di dati che fanno parte del gruppo di tablespace trasportabili. Facoltativamente, si possono specificare le tablespace (con il parametro TABLESPACES) e i proprietari degli oggetti (con il parametro OWNERS).

Terminata l'importazione, tutte le tablespace dell'insieme trasportabile rimangono nella modalità lettura-scrittura. Per collocare le nuove tablespace nella modalità lettura-scrittura, si può eseguire il comando ALTER TABLESPACE READ WRITE nel database di destinazione.

```
alter tablespace AGG_DATA read write;
alter tablespace AGG_INDEXES read write;
```

Si osservi che non è possibile modificare la proprietà degli oggetti trasportati.

Le tablespace trasportabili consentono lo spostamento molto rapido di database di grandi dimensioni. In una data warehouse, le tablespace trasportabili potrebbero servire a pubblicare gli aggregati da warehouse principali in data mart, o dai data mart a una data warehouse globale. I dati di sola lettura possono essere distribuiti velocemente in più database: invece di inviare script SQL, si possono inviare file di dati e metadati esportati. Questo processo di spostamento di dati modificati può semplificare enormemente le procedure per la gestione dei database remoti, dei data mart remoti e di operazioni di spostamento di dati onerose.

## TAN

**VEDERE ANCHE** ACOS; ASIN; ATAN; ATAN2; COS; COSH; NUMERI, FUNZIONI; SIN; TANH; Capitolo 8.

### SINTASSI

TAN(*valore*)

**DESCRIZIONE** TAN restituisce la tangente del *valore* di un angolo, espresso in radianti.

### ESEMPIO

```
select TAN(135*3.141593/180) Tan -- tangente di 135 gradi in radianti from DUAL;
```

## TANH

**VEDERE ANCHE** ACOS; ASIN; ATAN; ATAN2; COS; COSH; NUMERI, FUNZIONI; SIN; TAN; Capitolo 8.

### SINTASSI

TANH(*valore*)

**DESCRIZIONE** TANH restituisce la tangente iperbolica del *valore* di un angolo.

## TEMA

Un tema è una particolare tipologia di documento di testo. Le voci di testo possono contenere temi che non compaiono come parole nel documento stesso. Per esempio, un documento riguardante un mutuo può avere come tema la parola “banca” anche se questa non appare nel documento. Vedere il Capitolo 24.

## TERMOUT (SQL\*PLUS)

*Vedere SET.*

## TIME (SQL\*PLUS)

*Vedere SET.*

## TIMESTAMP

Il tipo di dati TIMESTAMP (*precisione\_secondo\_frazionario*) registra i valori anno, mese e giorno delle date, così come i valori ora, minuto e secondi degli orari, in cui *precisione\_secondo\_frazionario* è il numero di cifre contenute nella parte frazionaria del campo dei secondi della data/ora. I valori validi per *precisione\_secondo\_frazionario* sono compresi tra 0 e 9. Il valore predefinito è 6. vedere DATI, TIPI e il Capitolo 9.

## **TIMESTAMP WITH LOCAL TIME ZONE**

Il tipo di dati TIMESTAMP (*precisione\_secondo\_frazionario*) WITH LOCAL TIME ZONE registra i valori di TIMESTAMP WITH TIME ZONE, normalizzati al fuso orario del database nel momento in cui i dati vengono memorizzati. Quando recuperano i dati, gli utenti li visualizzeranno nel fuso orario della sessione.

## **TIMESTAMP WITH TIME ZONE**

Il tipo di dati TIMESTAMP (*precisione\_secondo\_frazionario*) WITH TIME ZONE registra i valori anno, mese e giorno delle date, così come i valori ora, minuto e secondi degli orari, in cui *precisione\_secondo\_frazionario* è il numero di cifre contenute nella parte frazionaria del campo dei secondi della data/ora. I valori validi per *precisione\_secondo\_frazionario* sono compresi tra 0 e 9. Il valore predefinito è 6

## **TIMING (Forma 1, SQL\*PLUS)**

Vedere SET.

## **TIMING (Forma 2, SQL\*PLUS)**

**VEDERE ANCHE** CLEAR, SET.

### **SINTASSI**

TIMI[NG] [ START *area* | STOP | SHOW ];

**DESCRIZIONE** TIMING tiene traccia del tempo trascorso da START a STOP tramite un nome di area. START apre un'area di temporizzazione e le assegna come titolo *area*. Il titolo deve essere di un'unica parola. Possono esistere contemporaneamente più aree: quella creata più di recente rimane l'area di temporizzazione corrente finché non viene cancellata. In quel momento, l'area immediatamente precedente diventa quella corrente. Questo significa che le aree di temporizzazione sono annidate. L'area più recente mostrerà sempre il valore di tempo minimo, perché il tempo trascorso delle aree precedenti sarà, per definizione, più lungo. Il tempo totale di qualunque area di temporizzazione è il suo tempo di rete più quelli di tutte le aree create dopo di lei.

SHOW fornisce il nome dell'area di temporizzazione corrente e il tempo trascorso.

STOP fornisce il nome dell'area di temporizzazione corrente e il tempo trascorso, cancella l'area e rende corrente l'area immediatamente precedente (se esiste). Per dettagli sul significato preciso del tempo nel proprio computer, si *consulti SQL\*Plus User's Guide and Reference* del proprio sistema operativo.

### **ESEMPIO**

Per creare un'area di temporizzazione di nome 1 si digiti:

```
timing start 1
```

Per visualizzare il nome dell'area corrente e il tempo trascorso senza interrompere il cronometraggio si digiti:

```
timing show
```

Per vedere il nome dell'area di temporizzazione corrente e il tempo trascorso chiudendo il cronometraggio e rendendo corrente l'area precedente si digit:

```
timing stop
```

## TIPO DI DATI LONG

Vedere DATI, TIPI

### TO\_CHAR (formati carattere)

**VEDERE ANCHE** TO\_CHAR (formati di data e numerici).

#### SINTASSI

```
TO_CHAR ( ncar | clob | nclob )
```

**DESCRIZIONE** La funzione TO\_CHAR (*carattere*) converte i dati NCHAR, NVARCHAR2, CLOB o NCLOB nel set di caratteri del database.

#### ESEMPIO

```
SELECT TO_CHAR(Titolo) FROM BIBLIOTECA;
```

### TO\_CHAR (formati di data e numerici)

**VEDERE ANCHE** DATE, FORMATI; DATE, FUNZIONI; NUMERICI, FORMATI; TO\_DATE; Capitolo 9.

#### SINTASSI

```
TO_CHAR(data, 'formato')  
TO_CHAR(numero, 'formato')
```

**DESCRIZIONE** TO\_CHAR riformatta un numero o una data in base a *formato*. *data* dev'essere una colonna definita come il tipo di dati DATE di Oracle. Non può essere una stringa, neppure se è nel formato standard (DD-MON-YY). L'unico modo di utilizzare una stringa in cui *data* compare nella funzione TO\_CHAR consiste nel racchiuderla in una funzione TO\_DATE. I formati disponibili per questa funzione sono numerosi. Sono elencati alle voci NUMERICI, FORMATI; e DATE, FORMATI.

#### ESEMPIO

Si osservi il formato dell'esempio seguente. È necessario operare in questo modo per evitare che SQL\*PLUS utilizzi il formato di data predefinito di TO\_CHAR che è lungo circa 100 caratteri:

```
column Formatted format a30 word_wrapped heading 'Formatted'  
  
select DataNascita, TO_CHAR(DataNascita,"MM/DD/YY") Formattata  
  from COMPLEANNO  
 where Nome = 'VICTORIA';  
  
DATANASCITA Formattata  
-----  
20-MAY-49  20/05/49
```

## TO\_CLOB

**VEDERE ANCHE** Capitoli 18 e 32.

### SINTASSI

TO\_CLOB ( *colonna\_lob* | *car* )

**DESCRIZIONE** TO\_CLOB converte i valori NCLOB in una colonna di tipo LOB o altre stringhe di caratteri in valori CLOB. *car* può essere uno qualsiasi tra i tipi di dati CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB o NCLOB. Oracle esegue questa funzione convertendo i dati del LOB di base dal set di caratteri nazionali al set di caratteri del database. Si osservi che i valori LONG possono essere convertiti direttamente in colonne di tipo CLOB tramite il comando ALTER TABLE.

## TO\_DATE

**VEDERE ANCHE** DATE,FORMATI; DATE, FUNZIONI; TO\_CHAR; Capitolo 8.

### SINTASSI

TO\_DATE(*stringa*, '*formato*' )

**DESCRIZIONE** TO\_DATE converte una *stringa* il cui formato è quello specificato in una data Oracle. Questa funzione accetta anche un numero al posto della stringa, ma con alcune limitazioni. *stringa* è una stringa letterale, un numero letterale o una colonna di database che contiene un numero o una stringa. In tutti i casi, tranne uno, il formato deve corrispondere a quello descritto da *formato*. Solo se la stringa è nel formato ‘DD-MON-YY’ si può omettere il formato. Si osservi che *formato* è limitato. Per un elenco di formati accettabili per TO\_DATE, si consulti DATE, FORMATI.

### ESEMPIO

```
select TO_DATE('02/22/02','MM/DD/YY') from DUAL;
TO_DATE(
-----
22.02.02
```

## TO\_DSINTERVAL

**VEDERE ANCHE** DATI, TIPI.

### SINTASSI

TO\_DSINTERVAL ( *car* [ '*param\_nls*' ] )

**DESCRIZIONE** TO\_DSINTERVAL converte una stringa del tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un tipo INTERVAL DAY TO SECOND. *car* è la stringa di caratteri da convertire L'unico parametro NLS valido che si può specificare in questa funzione è NLS\_NUMERIC\_CHARACTERS. Questo argomento può avere come forma: NLS\_NUMERIC\_CHARACTERS = “dg” dove *d* e *g* rappresentano rispettivamente il carattere decimale e il separatore di gruppo.

## TO\_LOB

**VEDERE ANCHE** INSERT, Capitolo 32.

### SINTASSI

TO\_LOB(*colonna\_long*)

**DESCRIZIONE** TO\_LOB converte i valori LONG nella *colonna\_long* in valori LOB. Questa funzione può essere applicata solo a una colonna di tipo LONG, e solo nel select list di una subquery in un comando insert.

## TO\_MULTI\_BYTE

**VEDERE ANCHE** CONVERSIONE, FUNZIONI; Capitolo 10.

### SINTASSI

TO\_MULTI\_BYTE(*stringa*)

**DESCRIZIONE** TO\_MULTI\_BYTE converte i caratteri a un byte di una *stringa* di caratteri ai loro equivalenti a più byte. Se un carattere non ha un equivalente multibyte la funzione restituisce il carattere senza modifiche.

## TO\_NCHAR (character)

**VEDERE ANCHE** CONVERSIONE, FUNZIONI; TRANSLATE ... USING.

### SINTASSI

TO\_NCHAR ( {*car* | *clob* | *nclob*} [, *fmt* [, 'param\_nls']] )

**DESCRIZIONE** TO\_NCHAR (*carattere*) converte una stringa di caratteri, CLOB o NCLOB dal set di caratteri del database nel set di caratteri nazionali. Questa funzione equivale alla funzione TRANSLATE ... USING con una clausola USING nel set di caratteri nazionali.

## TO\_NCHAR(data/ora)

**VEDERE ANCHE** CONVERSIONE, FUNZIONI.

### SINTASSI

TO\_NCHAR ( { *dataora* | *intervallo* } [, *fmt* [, 'param\_nls']] )

**DESCRIZIONE** TO\_NCHAR (*data/ora*) converte una stringa di caratteri del tipo di dati DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE, INTERVAL MONTH TO YEAR o INTERVAL DAY TO SECOND dal set di caratteri del database nel set di caratteri nazionali.

## TO\_NCHAR(numero)

**VEDERE ANCHE** CONVERSIONE, FUNZIONI.

**SINTASSI**

```
TO_NCHAR ( n [ , fmt [ , 'param_nls']] )
```

**DESCRIZIONE** TO\_NCHAR (*numero*) converte un numero in una stringa nel set di caratteri del tipo di dati NVARCHAR2. Le opzioni facoltative *fmt* e *param\_nls* corrispondenti a *n* possono essere del tipo di dati DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE, INTERVAL MONTH TO YEAR o INTERVAL DAY TO SECOND.

**TO\_NCLOB**

**VEDERE ANCHE** CONVERSIONE, FUNZIONI; Capitolo 32.

**SINTASSI**

```
TO_NCLOB ( colonna_lob | car )
```

**DESCRIZIONE** TO\_NCLOB converte i valori CLOB in una colonna di tipo LOB o altre stringhe di caratteri in valori NCLOB. *car* può essere un qualsiasi tipo di dati CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB o NCLOB. Oracle implementa questa funzione convertendo il set di caratteri della colonna di tipo LOB dal set di caratteri del database nel set di caratteri nazionali.

**TO\_NUMBER**

**VEDERE ANCHE** CONVERSIONE, FUNZIONI; Capitolo 10.

**SINTASSI**

```
TO_NUMBER(stringa [ , fmt [ , 'param_nls']] )
```

**DESCRIZIONE** TO\_NUMBER converte una *stringa* di caratteri in un tipo di dati numerico. Richiede che la stringa contenga un numero formattato correttamente e che siano presenti i soli caratteri da 0 a 9, i segni - e + e il punto decimale. Questa funzione è inutile, dato che Oracle esegue conversioni di data automatiche, tranne quando vengono utilizzate per colonne di caratteri contenenti numeri nelle clausole order by o nelle comparazioni.

**ESEMPIO**

```
TO_NUMBER('333.46')
```

**TO\_SINGLE\_BYTE**

**VEDERE ANCHE** CONVERSIONE, FUNZIONI; Capitolo 10.

**SINTASSI**

```
TO_SINGLE_BYTE(stringa)
```

**DESCRIZIONE** TO\_SINGLE\_BYTE converte i caratteri a più byte di una *stringa* di caratteri nei relativi equivalenti a byte singoli. Se un carattere multibyte non ha un equivalente monobyte, la funzione lo restituisce senza modificarlo.

## TO\_TIMESTAMP

**VEDERE ANCHE** CONVERSIONE, FUNZIONI; TO\_CHAR; TIMESTAMP; TO\_TIMESTAMP\_TZ.

### SINTASSI

```
TO_TIMESTAMP ( car [ . fmt ['param_nls']] )
```

**DESCRIZIONE** TO\_TIMESTAMP converte *car* del tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un valore del tipo di dati TIMESTAMP con un formato *fmt* se *car* non è nel formato predefinito del tipo di dati TIMESTAMP. In questa funzione, l'opzione facoltativa *param\_nls* ha lo stesso scopo che aveva nella funzione TO\_CHAR per la conversione delle date.

## TO\_TIMESTAMP\_TZ

**VEDERE ANCHE** CONVERSIONE, FUNZIONI; TO\_TIMESTAMP; TIMESTAMP WITH TIME ZONE.

### SINTASSI

```
TO_TIMESTAMP_TZ ( car [ . fmt ['param_nls']] )
```

**DESCRIZIONE** TO\_TIMESTAMP\_TZ converte *car* del tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un valore del tipo di dati TIMESTAMP WITH TIMEZONE.

## TO\_YMINTERVAL

**VEDERE ANCHE** CONVERSIONE, FUNZIONI; INTERVAL YEAR TO MONTH.

### SINTASSI

```
TO_YMINTERVAL ( car )
```

**DESCRIZIONE** TO\_YMINTERVAL converte una stringa di caratteri del tipo di dati CHAR, VARCHAR2, NCHAR o NVARCHAR2 in un tipo INTERVAL YEAR TO MONTH, in cui *car* è la stringa di caratteri da convertire.

## TRANSAZIONE

Una transazione è una sequenza di istruzioni SQL che Oracle tratta come un'unica entità. L'insieme di modifiche viene reso permanente tramite l'istruzione COMMIT. Gli effetti di una transazione possono essere annullati in toto o in parte attraverso l'istruzione ROLLBACK.

Oracle gestisce le transazioni sia con meccanismi di blocco (*vedere* LOCK) sia con un *modello di coerenza multiversione*, che in sostanza si comporta come se ogni transazione disponesse di una propria copia del database, ossia crea più versioni concorrenti dello stesso database. Una transazione inizia con l'esecuzione della prima istruzione SQL che la riguarda e termina con le istruzioni COMMIT o ROLLBACK. Per default, Oracle garantisce che le transazioni dispongano di una *coerenza di lettura a livello di istruzione*, ovvero quando si esegue una query i dati rimangono congelati mentre Oracle li raccoglie e li restituisce al programma. Se però una transazione ha più query, ciascuna è coerente con se stessa ma non è detto che lo sia con le altre.

Se si desidera mantenere una visuale coerente dei dati durante un'intera transazione, questa deve utilizzare un meccanismo di *coerenza di lettura a livello di transazione* che garantisce che una transazione non risentirà degli effetti del salvataggio di altre transazioni concorrenti. Queste transazioni di sola lettura, riconoscibili perché iniziano con l'istruzione SET TRANSACTION READ ONLY, possono eseguire solo query e alcuni comandi di controllo (*vedere SET TRANSACTION*).

## TRANSLATE

**VEDERE ANCHE** REPLACE, Capitolo 10.

### SINTASSI

TRANSLATE(*stringa*,*da*,*a*)

**DESCRIZIONE** TRANSLATE esamina ogni carattere in *stringa*, poi controlla *da* per vedere se il carattere è presente nella stringa. In caso affermativo, TRANSLATE annota la posizione in cui è stato trovato il carattere in *da* e guarda nella stessa posizione di *a*. Qualsiasi carattere trovi in questa posizione, viene sostituito a quello di *stringa*.

### ESEMPIO

```
select TRANSLATE('I LOVE MY OLD THESAURUS','AEIOUY','123456')
      from DUAL;
      TRANSLATE('ILOVEMYOLDTHESAURUS')
-----
3 L4V2 M6 4LD TH2S15R5
```

## TRANSLATE ... USING

**VEDERE ANCHE** CONVERSIONE, FUNZIONI, CONVERT, TO\_NCHAR, UNISTR.

### SINTASSI

TRANSLATE ( *testo* USING { CHAR\_CS | NCHAR\_CS } )

**DESCRIZIONE** TRANSLATE... USING converte testo nel set di caratteri specificato per le conversioni tra il set di caratteri del database e quello di caratteri nazionali. L'argomento *testo* è l'espressione da convertire.

Se si specifica l'argomento USING CHAR\_CS, *testo* viene convertito nel set di caratteri del database. Il tipo di dati dell'output è VARCHAR2.

Se invece si specifica l'argomento USING NCHAR\_CS, *testo* viene convertito nel set di caratteri nazionali. Il tipo di dati dell'output è NVARCHAR2.

Questa funzione è simile alla funzione CONVERT di Oracle, ma dev'essere impiegata al posto di CONVERT se il tipo di dati dell'input o dell'output utilizzato è CHAR o NVARCHAR2. Se l'input contiene codepoint UCS2 o caratteri come la barra contraria (\), è necessario utilizzare la funzione UNISTR.

## TREAT

**VEDERE ANCHE** DEREF, REF, Capitolo 33.

## SINTASSI

TREAT ( *espr* AS [REF] [*schema.*] *tipo* )

**DESCRIZIONE** TREAT modifica il tipo dichiarato di un'espressione.

Se il tipo dichiarato di *espr* è *tipo\_origine*, allora il tipo dovrà essere un qualche supertipo o sottotipo di *tipo\_origine*. Se il tipo più specifico di *espr* è *tipo* (o qualche sottotipo di *tipo*), TREAT restituisce *espr*. Se il tipo più specifico di *espr* non è *tipo* (o qualche sottotipo di *tipo*), TREAT restituisce NULL.

Se il tipo dichiarato di *espr* è *tipo\_origine* REF, allora *tipo* dev'essere qualche sottotipo o supertipo di *tipo\_origine*. Se il tipo più specifico di DEREF(*espr*) è *tipo* (o un sottotipo di *tipo*), TREAT restituisce *espr*. Se il tipo più specifico di DEREF(*espr*) non è *tipo* (o un sottotipo di *tipo*), allora TREAT restituisce NULL.

## TRIGGER

Un trigger di database è una stored procedure associata a una tabella che Oracle esegue automaticamente al verificarsi di uno o più eventi (prima, dopo o al posto di un inserimento, un aggiornamento o una cancellazione) che la riguardano. I trigger possono essere eseguiti sull'intera tabella o singolarmente su ciascuna riga della tabella. Per una discussione completa sui trigger e relativi esempi, si rimanda al Capitolo 28, mentre per la sintassi si consulti la voce CREATE TRIGGER in questo capitolo.

## TRIM

**VEDERE ANCHE** LTRIM, RTRIM, Capitolo 7.

## SINTASSI

```
TRIM
( [{ { LEADING | TRAILING | BOTH } [caratteri_da_tagliare]
      | caratteri_da_tagliare
    }   FROM   ]  origine_da_tagliare )
```

**DESCRIZIONE** TRIM consente di tagliare i caratteri iniziali o finali (oppure entrambi) da una stringa di caratteri. Se *caratteri\_da\_tagliare* o *origine\_da\_tagliare* è un letterale, lo si deve racchiudere tra apici.

Se si specifica LEADING, Oracle rimuove qualsiasi carattere iniziale uguale a *caratteri\_da\_tagliare*. Se invece si specifica TRAILING, Oracle rimuove qualsiasi carattere finale uguale a *caratteri\_da\_tagliare*. Se si specifica BOTH o nessuna delle tre opzioni, Oracle rimuove i caratteri iniziali e finali uguali a *caratteri\_da\_tagliare*.

Se si specifica solo *sorgente\_da\_tagliare*, Oracle rimuove gli spazi vuoti iniziali e finali. Se *origine\_da\_tagliare* o *caratteri\_da\_tagliare* è un valore NULL, la funzione TRIM restituisce un valore NULL.

## ESEMPIO

```
select TRIM('' from Titolo) from RIVISTA;
select TRIM(leading '' from Titolo) from RIVISTA;
select TRIM(trailing '' from Titolo) from RIVISTA;
```

## TRIMOUT (SQL\*PLUS)

Vedere SET.

## TRUNC (Forma 1, per date)

**VEDERE ANCHE** COLUMN; DATE, FUNZIONI; TRUNC (Forma 2, per numeri); Capitolo 9.

### SINTASSI

TRUNC(*data*, 'formato')

**DESCRIZIONE** TRUNC esegue il troncamento di *data* in base al valore di *formato*. Se non si specifica questo parametro, *data* viene troncata alla mezzanotte del giorno in corso, per qualsiasi ora fino a, e inclusa, 11:59:59 P.M. (subito prima di mezzanotte). Per i valori validi per questo *formato* di data, si consulti la voce DATE, FORMATI.

Il risultato di TRUNC è sempre una data con l'ora impostata a mezzanotte, l'inizio di una giornata.

## TRUNC (Forma 2, per numeri)

**VEDERE ANCHE** COLUMN; NUMERI, FUNZIONI; TRUNC (Forma 1, per date); Capitolo 8.

### SINTASSI

TRUNC(*valore*,*precisione*)

**DESCRIZIONE** TRUNC è il valore troncato alla precisione desiderata.

### ESEMPI

```
TRUNC(123.45,0) = 123
TRUNC(123.45,-1) = 120
TRUNC(123.45,-2) = 100
```

## TRUNCATE

**VEDERE ANCHE** CREATE CLUSTER, DELETE, DROP TABLE, TRIGGER, Capitolo 18.

### SINTASSI

```
TRUNCATE
{ TABLE [schema .] tabella [{ PRESERVE | PURGE } MATERIALIZED VIEW LOG]
| CLUSTER [schema .] cluster }
[ { DROP | REUSE } STORAGE];
```

**DESCRIZIONE** TRUNCATE rimuove tutte le righe da una tabella o da un cluster. È possibile operare solo su cluster indicizzati, e non su cluster hash, (vedere CREATE CLUSTER). Se si aggiunge l'opzione DROP STORAGE, TRUNCATE annulla l'allocazione dello spazio relativo alle righe eliminate; se invece si aggiunge l'opzione REUSE STORAGE, TRUNCATE mantiene lo spazio allocato per le nuove righe della tabella. DROP STORAGE è l'impostazione predefinita.

Il comando TRUNCATE è più veloce di un comando DELETE, poiché non genera informazioni di rollback, non attiva trigger DELETE (quindi deve essere utilizzato con cautela) e non

registra nulla in un log di viste materializzate. Inoltre, l'uso di TRUNCATE non invalida gli oggetti che dipendono dalle righe cancellate o i privilegi esistenti sulla tabella.

**NOTA** *Non è possibile effettuare il rollback di un'istruzione TRUNCATE.*

## TTITLE

**VEDERE ANCHE** ACCEPT, BTITLE, DEFINE, PARAMETRI, REPFOOTER, REPHEADER, Capitolo 14.

### SINTASSI

TTI[TLE] [*opzione* [*testo|variabile*]... | OFF | ON]

**DESCRIZIONE** TTITLE inserisce un titolo (che può anche occupare più di una linea) all'inizio di ogni pagina di un report. OFF e ON rispettivamente eliminano e ripristinano la visualizzazione del testo senza modificarne i contenuti. TTITLE da solo visualizza le opzioni correnti e *testo* o *variabile*.

*testo* è il titolo che si desidera assegnare a questo report, mentre *variabile* è una variabile definita dall'utente o una gestita a livello di sistema come SQL.LNO (numero di linea corrente), SQL.PNO (numero di pagina corrente), SQL.RELEASE (numero della release corrente di Oracle), SQL.SQLCODE (codice di errore corrente) e SQL.USER (nome utente).

SQL\*PLUS utilizza ttitle nella sua nuova forma se riconosce nella prima parola dopo ttitle un'opzione valida. Le opzioni valide sono:

- COL *n* salta direttamente alla posizione *n* (dal margine sinistro) della linea corrente.
- S[KIP] *n* visualizza *n* linee vuote. Se non viene specificato alcun numero, viene stampata una sola linea vuota. Se *n* vale 0, non viene visualizzata alcuna linea vuota e la posizione di stampa corrente diventa la prima della linea corrente (all'estremità sinistra della pagina).
- TAB *n* salta di *n* tabulazioni verso destra (verso sinistra se *n* è negativo).
- BOLD stampa i dati dell'output in grassetto.
- LE[FT], CE[NTER] e R[IGHT] giustificano a sinistra, centrano o giustificano a destra i dati della linea corrente. Eventuali stringhe di testo e variabili che seguissero questi comandi verrebbero giustificate anch'esse, fino alla fine del comando. CENTER e RIGHT utilizzano il valore impostato da SET LINESIZE per determinare dove disporre il testo e le variabili.
- FORMAT stringa specifica il modello del formato che controlla la visualizzazione del testo o delle variabili seguenti e segue la stessa sintassi di FORMAT in un comando COLUMN; per esempio, FORMAT A12 o FORMAT \$999,990.99. Ogni volta che compare FORMAT, esso scavalca il precedente formato attivo fino a quel momento. Se non si specifica alcun modello per FORMAT, viene utilizzato quello impostato da SET NUMFORMAT. Se NUMFORMAT non è stato definito, viene utilizzato il modello predefinito per SQL\*PLUS.

Le date vengono visualizzate in base al formato predefinito, a meno che la data di una variabile non sia stata convertita mediante la funzione TO\_CHAR.

In un comando title può essere utilizzato un numero qualsiasi di opzioni, blocchi di testo e variabili. Ognuno viene stampato nell'ordine specificato e viene formattato in base alle clausole che lo precedono.

## TUPLA

È un sinonimo di riga.

## TYPE

Vedere TIPO DI DATI ASTRATTO e CREATE TYPE.

### TYPE (Embedded SQL)

**VEDERE ANCHE** Guida di programmazione del precompilatore.

#### SINTASSI

```
EXEC SQL TYPE tipo IS tipo_dati
```

**DESCRIZIONE** PL/SQL consente di assegnare un tipo di dati esterno di Oracle a un tipo di dati definito dall'utente. Questo può contenere una lunghezza, una precisione o una scala. Questo tipo di dati esterno viene reso equivalente a quello definito dall'utente e viene assegnato a tutte le variabili host di quel tipo. Per un elenco dei tipi di dati esterni, si consulti la guida di programmazione del precompilatore.

## TYPE BODY

Vedere TIPO DI DATI ASTRATTO, CREATE TYPE, e METODO.

### TZ\_OFFSET

**VEDERE ANCHE** SESSIONTIMEZONE, DBTIMEZONE.

#### SINTASSI

```
TZ_OFFSET
( { ' nome_fuso_orario'
  | '{ + | - } hh : mi'
  | SESSIONTIMEZONE
  | DBTIMEZONE
  })
```

**DESCRIZIONE** TZ\_OFFSET restituisce la differenza di fuso orario corrispondente al valore inserito in base alla data di esecuzione dell'istruzione. Si può specificare un nome di fuso orario valido, una differenza di fuso orario da UTC (che restituisce se stessa) oppure le parole chiave SESSIONTIMEZONE o DBTIMEZONE. Per un elenco dei valori validi, si esegua una query sulla colonna TZNAME della vista a prestazioni dinamiche V\$TIMEZONE\_NAMES.

## UID

**VEDERE ANCHE** SESSION.

#### SINTASSI

UID

**DESCRIZIONE** UID restituisce un valore intero che identifica in modo univoco l'utente della sessione (l'utente connesso).

## UNDEFINE

**VEDERE ANCHE** ACCEPT, DEFINE, PARAMETRI, Capitoli 14 e 16.

### SINTASSI

UNDEF[INE] *variabile*

**DESCRIZIONE** UNDEFINE rimuove la definizione di una variabile utente definita da ACCEPT, DEFINE o come parametro del comando START. Si possono eliminare le definizioni dei valori correnti di più variabili con un solo comando. Il formato è mostrato nel listato seguente:

```
undefine variable1 variable2 ...
```

### ESEMPIO

Per eliminare la definizione della variabile di nome Total, si digit:

```
undefine Total
```

## UNDERLINE (SQL\*PLUS)

*Vedere SET.*

## UNION

**VEDERE ANCHE** INTERSECT, MINUS, OPERATORI DI QUERY, Capitolo 12.

### SINTASSI

```
select...  
  UNION [ALL]  
select...
```

**DESCRIZIONE** UNION combina due query. Restituisce tutte le righe distinte di entrambe le istruzioni select oppure, qualora sia indicata l'opzione ALL, tutte le righe senza controllarne l'unicità (e quindi comprendendo eventuali doppioni). Il numero di colonne e i tipi di dati devono essere identici tra le due istruzioni select, condizione non richiesta per i nomi delle colonne. Per una discussione riguardante le differenze importanti e gli effetti di INTERSECT, UNION e MINUS, e il ruolo che la precedenza ha nei risultati, si rimanda Capitolo 12.

## UNISTR

**VEDERE ANCHE** CONVERSIONE, FUNZIONI, CONVERT, TRANSLATE...USING.

### SINTASSI

```
UNISTR ( "stringa" )
```

**DESCRIZIONE** UNISTR accetta come proprio argomento una stringa in un qualsiasi set di caratteri e la restituisce nel formato UNICODE del set di caratteri UNICODE del database. Per includere nella stringa i caratteri codepoint UCS2, si utilizzi la barra contraria di escape (\) seguita dal numero successivo. Per includere la stessa barra contraria, la si faccia precedere da un'altra barra rovesciata (\|).

Questa funzione assomiglia alla funzione TRANSLATE ... USING, con l'unica differenza che UNISTR offre il carattere di escape per i codepoint UCS2 e per le barre contrarie.

## UNITÀ DI LAVORO

In Oracle, una transazione equivale a un'unità di lavoro logica e comprende tutte le istruzioni SQL da quando ci si è collegati, si è salvato per l'ultima volta o si sono annullati i cambiamenti. Per questo motivo, una transazione può racchiudere una o più istruzioni SQL.

## UNITÀ LOGICHE DI LAVORO

*Vedere* TRANSAZIONE.

## UPDATE (Forma 1, Embedded SQL)

**VEDERE ANCHE** EXECUTE IMMEDIATE, FOR, PREPARE, SELECT (Forma 2), guida di programmazione del precompilatore.

### SINTASSI

```
EXEC SQL [AT { dbnome | :variabile_host }]
[FOR { :intero_host | intero }]
UPDATE
[ (subquery)
| [schema .] { tabella | vista } [ @db_link | PARTITION (nome_part) ] ]
SET
{ colonna = { espr | (subquery_2) }
| (colonna [, colonna]...) = (subquery_1) }
[, { colonna = { espr | (subquery_2) }
| (colonna [, colonna]...) = (subquery_1) } ]...
[WHERE { condition | CURRENT OF cursore }]
[{ RETURN | RETURNING } espr [, espr]... INTO
:variabile_host [[INDICATOR] :variabile_ind]
[, :variabile_host [[INDICATOR] :variabile_ind]]...]
```

**DESCRIZIONE** *Vedere* la descrizione delle varie clausole in UPDATE (Forma 3). Gli elementi tipici di Embedded SQL sono i seguenti:

- AT dbnome, che facoltativamente assegna a un database aperto precedentemente da un'istruzione CONNECT il nome di un database preso da una precedente istruzione DECLARE DATABASE.
- FOR :intero\_host, che imposta il numero massimo di righe da recuperare. *intero* è il nome di una variabile host.
- *espr* può comprendere una variabile host :variabile[:indicatore].
- La clausola where può contenere variabili o array host.
- CURRENT OF aggiorna l'ultima riga recuperata del *cursore* specificato. Tuttavia, il cursore dev'essere aperto e posizionato sulla riga. In caso contrario, CURRENT OF, che fa parte della clausola where, fa sì che quest'ultima non trovi alcuna riga da poter aggiornare. Il cursore dev'essere stato dichiarato precedentemente in un'istruzione DECLARE CURSOR mediante select...for update of.

Se una variabile host di set o where è un array, allora tutte le variabili host di set e where devono essere array, anche se non necessariamente della stessa dimensione. Se si tratta di array,

update viene eseguito una volta per ciascun gruppo di componenti dell'array, e può aggiornare zero o più righe. Il numero massimo dipende dalla dimensione dell'array più piccolo o dal valore intero della clausola for, se ne è stato specificato uno. Per ulteriori dettagli si consulti FOR.

## UPDATE (Forma 2, PL/SQL)

**VEDERE ANCHE** DECLARE CURSOR, SUBQUERY.

### SINTASSI

```
UPDATE [utente.]tabella[@dblink]
SET { colonna = espressione | colonna = (select espressione...)
[ [,colonna = espressione]... | 
[,colonna = (select espressione...)...] | 
(colonna [,colonna]...) = (subquery)}
[WHERE {condizione | CURRENT OF cursor}];
```

**DESCRIZIONE** Il comando UPDATE di PL/SQL funziona allo stesso modo dell'istruzione UPDATE di SQL: l'unica differenza è la clausola where alternativa WHERE CURRENT OF cursor. In questo caso, l'istruzione update modifica solo la riga su cui si è spostato attualmente il cursore a seguito dell'ultimo FETCH. L'istruzione select del cursore deve includere le parole FOR UPDATE.

Come insert, delete e select...into, anche l'istruzione update utilizza sempre il cursore implicito di nome SQL. Quest'ultimo non viene mai dichiarato (anche se l'istruzione select...for update deve essere dichiarata in un cursore affinché WHERE CURRENT OF cursor possa essere utilizzata). I suoi attributi sono impostati nel modo seguente:

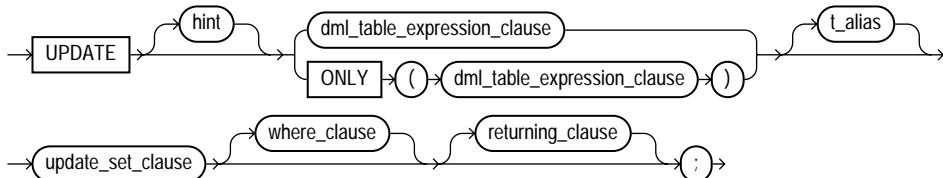
- SQL%ISOPEN è irrilevante.
- SQL%FOUND è TRUE se sono state aggiornate una o più righe, FALSE se nessuna riga è stata aggiornata. SQL%NOTFOUND è l'inverso di SQL%FOUND.
- SQL%ROWCOUNT contiene il numero di righe aggiornate.

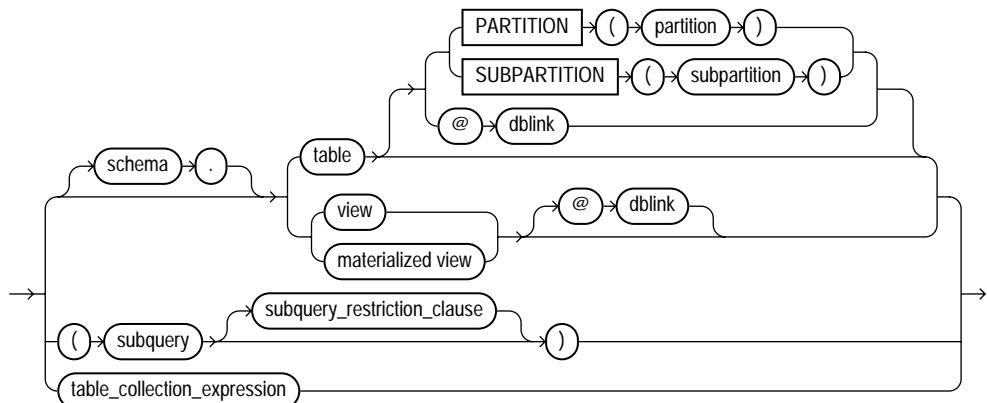
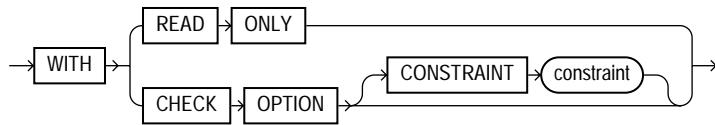
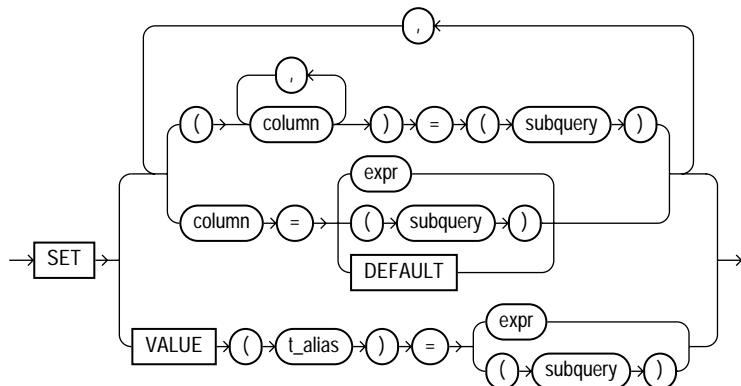
## UPDATE (Forma 3, comando SQL)

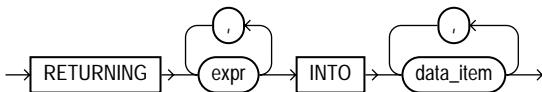
**VEDERE ANCHE** DELETE, INSERT, SELECT, SUBQUERY, WHERE, Capitolo 15.

### SINTASSI

update



**DML\_table\_expression\_clause****subquery\_restriction\_clause****table\_collection\_expression****update\_set\_clause****where\_clause**

**returning\_clause**

**DESCRIZIONE** UPDATE aggiorna (modifica) i valori delle colonne elencate nella tabella specificata. La clausola where può contenere una subquery correlata. Una subquery può eseguire selezioni dalla tabella aggiornata, anche se deve restituire solo una riga. Senza alcuna opzione where, vengono aggiornate tutte le righe. Con una clausola where, vengono aggiornate solo le righe selezionate. Le espressioni vengono valutate all'esecuzione del comando e i loro risultati sostituiscono i valori correnti delle colonne nelle righe.

L'eventuale subquery deve selezionare lo stesso numero di colonne (con tipi di dati compatibili) di quelle racchiuse tra parentesi sul lato sinistro dell'opzione set. Le colonne impostate uguali a una certa espressione possono precedere le colonne tra parentesi impostate uguali a una certa subquery, tutte all'interno di un'unica istruzione update.

**ESEMPIO**

Per impostare a NULL tutti i valori di Editore:

```
update BIBLIOTECA set Editore = NULL ;
```

Il codice seguente aggiorna la città di Walpole nella tabella COMFORT impostando a NULL tutti i valori Precipitazione, la temperatura massima uguale a quella di Manchester e quella minima pari alla massima meno 10 gradi.

```
update COMFORT set Precipitazione = NULL,
    (Mezzogiorno, Mezzanotte) =
    (select Temperatura, Temperatura - 10
     from CLIMA
     where Citta = 'MANCHESTER')
    where Citta = 'WALPOLE';
```

**UPPER**

**VEDERE ANCHE** LOWER, Capitolo 7.

**SINTASSI**

```
UPPER(stringa)
```

**DESCRIZIONE** UPPER converte ogni lettera di una *stringa* in maiuscolo.

**ESEMPIO**

```
upper('Guarda che hai fatto!')
```

produce:

```
GUARDA CHE HAI FATTO!
```

**UROWID**

UROWID è il tipo di dati utilizzato per le chiavi primarie delle tabelle di solo indice, conosciuto anche come Universal RowID. La dimensione massima e quella predefinita sono di 4.000 byte. Vedere DATI, TIPI e CREATE TABLE.

## USER

**VEDERE ANCHE** UID.

### SINTASSI

USER

**DESCRIZIONE** *User* è il nome attraverso cui l'utente corrente è riconosciuto da Oracle. *User* è una funzione, e come tale può essere sottoposta a query in qualsiasi istruzione SELECT.

### ESEMPIO

```
select USER from DUAL;
```

## USERENV

**VEDERE ANCHE** CARATTERI, FUNZIONI.

### SINTASSI

USERENV(*opzione*)

**DESCRIZIONE** USERENV è una funzione tradizionale, sostituita dallo spazio di nomi UserEnv di SYS\_CONTEXT. Le opzioni e i relativi valori di restituzione sono i seguenti.

'CLIENT_INFO'	CLIENT_INFO restituisce fino a 64 byte di informazioni sulla sessione dell'utente che possono essere memorizzati da un'applicazione che usa il package DBMS_APPLICATION_INFO.
'ENTRYID'	ENTRYID restituisce l'identificativo della voce di audit disponibile. Questo attributo non può essere utilizzato in istruzioni SQL distribuite. Per usare questa parola chiave in USERENV, è necessario impostare a TRUE il parametro di inizializzazione AUDIT_TRAIL.
"INSTANCE"	INSTANCE restituisce il numero di identificazione dell'istanza corrente.
'ISDBA'	ISDBA restituisce 'TRUE' se l'utente è stato autenticato con privilegi DBA, tramite il sistema operativo oppure tramite un password file.
'LANG'	LANG restituisce l'abbreviazione ISO per il nome della lingua, una forma più breve rispetto al parametro 'LANGUAGE'.
'LANGUAGE'	LANGUAGE restituisce la lingua e il territorio utilizzati attualmente dalla sessione insieme al set di caratteri del database in questa forma: language_territory.characterset
'SESSIONID'	SESSIONID restituisce l'identificativo di sessione di audit. Questo attributo non può essere utilizzato in istruzioni SQL distribuite.
'TERMINAL'	TERMINAL restituisce l'identificativo del sistema operativo per il terminale della sessione corrente. Nelle istruzioni SQL distribuite, questo attributo restituisce l'identificativo della sessione locale. In un ambiente distribuito, questo attributo è supportato solo per le istruzioni SELECT remote, e non per le operazioni remote di inserimento, aggiornamento o cancellazione.

## USERNAME

*username* è una parola che identifica un utente autorizzato del sistema operativo installato nel computer host o di Oracle. A ciascun username è sempre associata una password.

## VALORE PREDEFINITO

Il valore predefinito è un valore che viene utilizzato se non ne viene specificato o inserito esplicitamente un altro.

## VALUE

**VEDERE ANCHE** DEREF, REF, Capitolo 33.

### SINTASSI

VALUE ( *variabile\_correlazione* )

**DESCRIZIONE** In un’istruzione SQL, VALUE accetta come proprio argomento una variabile di correlazione (alias di tabella) associata a una riga di una tabella oggetto e restituisce le istanze dell’oggetto memorizzate in questa tabella oggetto. Il tipo delle istanze dell’oggetto è il medesimo di quello della tabella oggetto. Vedere il Capitolo 33.

## VAR (Embedded SQL)

**VEDERE ANCHE** Vedere SELECT (Forma 2); VARIABILI, DICHIARAZIONE.

### SINTASSI

```
EXEC SQL VAR variabile_host
{ IS dtyp [({ lunghezza | precisione, scala })]
 [CONVBUFSZ [IS] (dimensione)]
 | CONVBUFSZ [IS] (dimensione) }
```

**DESCRIZIONE** PL/SQL permette di sovrascrivere l’assegnamento del tipo di dati predefinito di una variabile tramite il comando VAR. Una volta dichiarata, una variabile utilizza il tipo di dati assegnato nel comando declare. VAR consente di modificare il tipo di dati di una variabile dichiarata all’interno di un blocco PL/SQL.

## VAR\_POP

**VEDERE ANCHE** FUNZIONI DI GRUPPO, VAR\_SAMP, VARIANCE.

### SINTASSI

VAR\_POP ( *espr* ) [OVER ( *clausola\_analitica* )]

**DESCRIZIONE** VAR\_POP restituisce la varianza della popolazione di un insieme di numeri dopo aver eliminato gli eventuali valori NULL contenuti in questo insieme.

## VAR\_SAMP

**VEDERE ANCHE** FUNZIONI DI GRUPPO, VAR\_POP, VARIANCE.

### SINTASSI

VAR\_SAMP ( *espr* ) [OVER ( *clausola\_analitica* )]

**DESCRIZIONE** VAR\_SAMP restituisce la varianza del campione di un insieme di numeri dopo aver eliminato gli eventuali valori NULL contenuti in questo insieme.

## VARCHAR2

*Vedere DATI, TIPI.*

## VARIABILE INDICATORE

**VEDERE ANCHE** : (due punti, prefisso della variabile host), guida di programmazione del precompilatore.

### SINTASSI

:*nome* [:INDICATOR] :*indicatore*

**DESCRIZIONE** Una variabile indicatore, utilizzata quando sia il nome sia l'indicatore sono nomi di variabili del linguaggio di host, è un campo di dati il cui valore indica se una variabile host dev'essere o meno considerata come NULL (la parola chiave INDICATOR è inserita solo per semplificare la lettura e non ha alcun effetto).

*nome* può essere qualsiasi nome dei dati di una variabile host, utilizzata nel programma di host e contenuta nella sezione delle dichiarazioni del precompilatore. *indicatore* è definito anch'esso nella sezione delle dichiarazioni del precompilatore ed equivale a un intero di due byte.

Pochi linguaggi procedurali consentono l'uso diretto di una variabile contenente un valore NULL o indefinito. Oracle e SQL richiedono invece alcune modifiche minime. Per far sì che i linguaggi per i quali Oracle ha sviluppato precompilatori supportino le variabili NULL, una variabile indicatore viene associata alla variabile host in modo che si comporti come un flag, indicando cioè se la variabile è pari a NULL oppure no. La variabile host e la sua indicatrice possono essere utilizzate e impostate separatamente nel linguaggio host ma risultano sempre concatenate sia in SQL che in PL/SQL.

### ESEMPIO

```
BEGIN
    select Titolo, NomeAutore into :Titolo, :NomeAutore:IndNomeAutore
    from BIBLIOTECA_AUTORE;

    IF :NomeAutore:IndNomeAutore IS NULL
        THEN :NomeAutore:IndNomeAutore := 'Nessun Autore'
    END IF;
END;
```

Si osservi che la verifica eseguita per controllare se la variabile è NULL, e la conseguente assegnazione della stringa 'Nessun Autore' se effettivamente il contenuto precedente era NULL vengono effettuate entrambe con il nome di variabile concatenata. PL/SQL è abbastanza intelligente da controllare l'indicatore nel primo caso e impostare il valore della variabile nel secondo. L'insieme delle due variabili viene trattato esattamente come un'unica colonna di Oracle. È bene fare attenzione ai seguenti aspetti:

- All'interno di qualsiasi singolo blocco PL/SQL una variabile host dev'essere sempre sola, oppure sempre concatenata alla propria variabile indicatore.

- In PL/SQL non è possibile fare riferimento a una variabile indicatore sola, mentre lo si può fare nel programma host.
- Quando si impostano le variabili host da un programma host (ma al di fuori di PL/SQL) occorre tenere presenti questi aspetti.
  - Se si imposta la variabile indicatore pari a -1, la variabile concatenata viene considerata da PL/SQL come se fosse NULL sia nei test logici sia nelle operazioni di inserimento e aggiornamento.
  - Se si imposta la variabile indicatore a un valore maggiore o uguale a 0, la variabile concatenata viene considerata uguale al valore della variabile host, e NOT NULL.
  - PL/SQL controlla il valore di tutte le variabili indicatore all'ingresso di un blocco, reimpostandole all'uscita.

Quando si caricano le variabili host concatenate all'interno di PL/SQL tramite un'istruzione SQL con una clausola INTO, vengono applicate le regole seguenti nella valutazione della variabile indicatore effettuata all'interno del programma host ma fuori da PL/SQL.

- La variabile indicatore vale -1 se il valore caricato dal database è NULL. Il valore della variabile host è indefinito e come tale dovrebbe essere trattato.
- La variabile indicatore vale 0 se il valore del database è stato caricato completamente e correttamente nella variabile host.
- La variabile indicatore contiene un valore maggiore di zero (pari alla vera lunghezza dei dati nella colonna del database) se è stato possibile caricare solo una parte dei dati nella variabile host. Questa conterrà una versione troncata del valore contenuto nella colonna del database.

## VARIABILI UTENTE

*Vedere ACCEPT, DEFINE, PARAMETRI.*

## VARIABILI, DICHIARAZIONE (PL/SQL)

### SINTASSI

```
variabile [CONSTANT]
{tipo | identificativo%TYPE | [utente.]tabella%ROWTYPE}
[NOT NULL]
[{DEFAULT | :=} espressione];
```

**DESCRIZIONE** PL/SQL permette di dichiarare variabili all'interno dei blocchi. Se si dichiara una variabile con l'opzione CONSTANT, è necessario inizializzarne contestualmente il valore e sarà impossibile modificarlo in seguito.

Il tipo della variabile può essere uno di quelli standard di PL/SQL (*vedere DATI, TIPI*), il tipo di un'altra variabile PL/SQL o quello di una colonna di database indicata tramite un identificativo o un ROWTYPE (*vedere %ROWTYPE*) che consente di riferirsi a un record corrispondente a una tabella di database.

Se si include la clausola NOT NULL nella dichiarazione, diventa impossibile assegnare un valore NULL alla variabile e anzi diventa indispensabile inizializzarla contestualmente. L'espressione di inizializzazione (che segue DEFAULT o il simbolo di assegnamento :=) è una qualsiasi espressione PL/SQL valida da cui risulti un valore del tipo dichiarato.

## VARIABLE

**VEDERE ANCHE** PRINT.

### SINTASSI

```
VAR[ABLE] [nome_variabile {NUMBER|CHAR|CHAR (n)}]
```

**DESCRIZIONE** VARIABLE dichiara una bind variable cui si può fare riferimento in PL/SQL. A ogni variabile viene assegnato un *nome* e un tipo (NUMBER o CHAR). Per le variabili di tipo CHAR è possibile specificare una lunghezza massima (*n*).

### ESEMPI

Nell'esempio seguente, viene creata una variabile di nome *bal* il cui valore viene impostato al risultato di una funzione.

```
variable bal NUMBER
begin
    :bal := CONTROLLO_SALDO('DORAH TALBOT');
end;
```

## VARIANCE

**VEDERE ANCHE** ACCEPT, COMPUTE, DEFINE, FUNZIONI DI GRUPPO, PARAMETERS, STDDEV, VAR\_POP, VAR\_SAMP, Capitolo 8.

### SINTASSI

```
VARIANCE ( [ DISTINCT | ALL ] expr ) [OVER ( clausola_analitica )]
```

**DESCRIZIONE** VARIANCE fornisce la varianza di tutti i *valori* provenienti da un gruppo di righe. Come le altre funzioni di gruppo, anche VARIANCE ignora eventuali valori NULL.

## VARRAY

VARRAY è la clausola del comando CREATE TYPE che indica al database il limite di voci dell'array variabile. Per esempi dettagliati riguardanti gli array variabili, si rimanda al Capitolo 31. Per informazioni di carattere sintattico, si consulti CREATE TYPE.

## VERIFY (SQL\*PLUS)

*Vedere* SET.

## VERSIONE DI REVISIONE

La versione di revisione è il secondo numero nella numerazione delle versioni del software Oracle. Per esempio, in Oracle versione 9.0.1, la versione di revisione è 0.

## VINCOLO DI INTEGRITÀ

Un vincolo di integrità è una regola che limita l'intervallo di valori consentiti per una colonna. Viene applicato su una colonna quando si crea la tabella. Per la sintassi, si rimanda alla voce VINCOLI di questo capitolo.

Se non si assegna un nome al vincolo, Oracle assegnerà un nome scritto nella forma SYS\_C<sub>n</sub>, in cui *n* è un valore intero. Solitamente, i nomi assegnati da Oracle cambiano durante le importazioni, cosa che non accade con i nomi assegnati dall'utente.

NULL permette la presenza di valori NULL. NOT NULL specifica che ogni riga deve contenere su questa colonna un valore non nullo.

UNIQUE impone che i valori della colonna siano unici. Una tabella può possedere solo un vincolo di tipo PRIMARY KEY. Se una colonna è UNIQUE non può essere dichiarata come PRIMARY KEY (dato che anche PRIMARY KEY impone l'univocità). Un indice impone il vincolo unico o di chiave primaria, mentre la clausola USING INDEX con le sue opzioni specifica le caratteristiche di memorizzazione di questo indice. Per ulteriori informazioni sulle opzioni, si consulti CREATE INDEX.

REFERENCES identifica la colonna corrente come chiave esterna da [utente.]tabella [(colonna)]. Se si omette colonna, Oracle suppone che il nome di utente.tabella sia lo stesso di quello della tabella corrente. Si osservi che quando REFERENCES viene utilizzato in un vincolo\_tabella (esaminato tra breve), questo deve essere preceduto da FOREIGN KEY. In questo caso, non è stato necessario utilizzarlo dato che si fa riferimento a una sola colonna; vincolo\_tabella può indicare diverse colonne FOREIGN KEY. ON DELETE CASCADE fa sì che Oracle gestisca automaticamente l'integrità referenziale rimuovendo le righe contenenti chiavi esterne nelle tabelle dipendenti quando si rimuove la riga delle chiavi primarie nella tabella principale.

CHECK assicura che il valore della colonna superi una condizione come quella che segue:

```
Amount number(12,2) CHECK (Amount >= 0)
```

*condizione* può essere qualsiasi espressione valida (che valga TRUE o FALSE). Può contenere funzioni, colonne appartenenti a tabelle e letterali.

La clausola EXCEPTIONS INTO specifica una tabella in cui Oracle inserisce le informazioni relative alle righe che violano un determinato vincolo di integrità. Questa tabella deve essere locale.

L'opzione DISABLE consente di disattivare il vincolo di integrità già all'atto della creazione. Quando il vincolo è disattivato, Oracle non lo impone automaticamente. In seguito sarà possibile attivare il vincolo con la clausola ENABLE di ALTER TABLE.

Si possono creare vincoli anche a livello di tabella. I vincoli di tabella sono identici a quelli di colonna, tranne per il fatto che un vincolo unico può fare riferimento a più colonne, per esempio nella dichiarazione di un gruppo di tre colonne come chiave primaria o esterna.

## VINCOLO DI TABELLA

Un vincolo di tabella è un vincolo di integrità che si applica a più colonne della stessa tabella. Vedere VINCOLO DI INTEGRITÀ.

## VIRTUALE, COLONNA

Una colonna virtuale è una colonna nel risultato di una query i cui valori vengono calcolati a partire da quelli di altre colonne.

## VISITA DI UN ALBERO

La visita di un albero è l'atto di elaborare uno per uno ogni nodo di un albero. Vedere CONNECT BY.

## VISTA

Una vista (un oggetto di database) è una rappresentazione logica di una tabella. Deriva da una tabella ma non dispone di un proprio spazio di memoria. Spesso può essere utilizzata allo stesso modo delle tabelle. *Vedere CREATE VIEW e il Capitolo 18.*

## VISTA MATERIALIZZATA

Le viste materializzate possono essere utilizzate per aggregare anticipatamente i dati e migliorare le prestazioni delle query. Quando si crea una vista materializzata, Oracle crea una tabella fisica per contenere i dati che solitamente verrebbero letti tramite una vista. Quando si crea una vista materializzata, si dovrà specificare la query base della vista, nonché un programma per gli aggiornamenti dei suoi dati. Quindi si potrà indicizzare la vista materializzata per migliorare le prestazioni delle query eseguite su di essa. Di conseguenza, si potranno fornire i dati agli utenti nel formato richiesto, indicizzati in modo corretto. *Vedere CREATE MATERIALIZED VIEW e il Capitolo 23.*

## VISTA OGGETTO

Una vista oggetto sovrappone tipi di dati astratti a tabelle relazionali già esistente. Le viste oggetto consentono di accedere ai dati di una tabella relazionale tramite i normali comandi SQL oppure tramite le strutture del suo tipo di dati astratto. Le viste oggetto mettono a disposizione un ponte tecnologico tra i database relazionali e quelli relazionali ad oggetti. Per esempi relativi alle viste oggetto, si rimanda ai Capitoli 30 e 33.

## VISTE DEL DIZIONARIO DATI

**VEDERE ANCHE** Capitolo 37.

**DESCRIZIONE** Le viste del dizionario dati di Oracle sono descritte nel Capitolo 37. Gran parte dell'elenco seguente, che descrive le viste disponibili nel dizionario dati, è stato generato selezionando i record dalla vista DICTIONARY in Oracle, tramite un account non DBA. Di seguito le viste sono elencate in ordine alfabetico; nel Capitolo 37 vengono invece elencate e descritte in base alla loro funzione.

NOME_TABELLA	COMMENTI
ALL_ALL_TABLES	Descrizione di tutti gli oggetti e di tutte le tabelle relazionali cui l'utente può accedere.
ALL_ARGUMENTS	Argomenti in un oggetto accessibili all'utente.
ALL_ASSOCIATIONS	Tutte le associazioni disponibili per l'utente.
ALL_AUDIT_POLICIES	Tutti i criteri di audit del database.
ALL_BASE_TABLE_MVIEWS	Tutte le viste materializzate con log(s) nel database che l'utente è in grado di visualizzare.
ALL_CATALOG	Tutte le tabelle, viste, sinonimi, sequenze accessibili all'utente.
ALL_CLUSTERS	Descrizione dei cluster accessibili all'utente.

(segue)

(continua)

NOME_TABELLA	COMMENTI
ALL_CLUSTER_HASH_EXPRESSIONS	Funzioni di hash per tutti i cluster accessibili.
ALL_COLL_TYPES	Descrizione di tipi di collezioni specificati accessibili all'utente.
ALL_COL_COMMENTS	Commenti su colonne di tabelle e viste accessibili.
ALL_COL_PRIVS	Concessioni su colonne per le quali l'utente è il beneficiario, il concedente, il proprietario o per cui il beneficiario è un ruolo abilitato o PUBLIC.
ALL_COL_PRIVS_MADE	Concessioni su colonne per le quali l'utente è il proprietario o il concedente.
ALL_COL_PRIVS_REC'D	Concessioni su colonne per le quali l'utente è beneficiario di un ruolo abilitato o PUBLIC.
ALL_CONSTRAINTS	Definizioni dei vincoli sulle tabelle accessibili.
ALL_CONS_COLUMNS	Informazioni relative alle colonne accessibili nelle definizioni dei vincoli.
ALL_CONTEXT	Descrizione di tutti gli spazi di nomi del contesto attivo nella sessione corrente.
ALL_DB_LINKS	Database link accessibili all'utente.
ALL_DEF_AUDIT_OPTS	Opzioni di audit per i nuovi oggetti creati.
ALL_DEPENDENCIES	Dipendenze da e verso gli oggetti accessibili all'utente.
ALL_DIMENSIONS	Descrizione degli oggetti di dimensionamento accessibili al DBA.
ALL_DIM_ATTRIBUTES	Rappresentazione della relazione tra un livello di dimensionamento e una colonna dipendente dalla funzionalità.
ALL_DIM_CHILD_OF	Rappresentazione di una relazione gerarchica di tipo 1:n tra una coppia di livelli in una dimensione.
ALL_DIM_HIERARCHIES	Rappresentazione di una gerarchia di dimensioni.
ALL_DIM_JOIN_KEY	Rappresentazione di un join tra due tabelle di dimensionamento.
ALL_DIM_LEVELS	Descrizione dei livelli di dimensionamento che il DBA è in grado di visualizzare.
ALL_DIM_LEVEL_KEY	Rappresentazione delle colonne di un livello di dimensionamento.
ALL_DIRECTORIES	Descrizione di tutte le directory accessibili all'utente.
ALL_ERRORS	Errori correnti negli oggetti memorizzati che l'utente ha il permesso di creare.
ALL_EXTERNAL_LOCATIONS	Descrizione delle locazioni delle tabelle esterne cui l'utente può accedere.
ALL_EXTERNAL_TABLES	Descrizione delle tabelle esterne accessibili all'utente.
ALL_HISTOGRAMS	Sinonimo per ALL_TAB_HISTOGRAMS.
ALL_INDEXES	Descrizioni degli indici sulle tabelle accessibili all'utente.
ALL_INDEXTYPES	Tutti i tipi di indici disponibili per l'utente.
ALL_INDEXTYPE_COMMENTS	Commenti per i tipi di indici definiti dall'utente.

(segue)

*(continua)*

NOME_TABELLA	COMMENTI
ALL_INDEXTYPE_OPERATORS	Tutti gli operatori disponibili per l'utente.
ALL_IND_COLUMNS	Colonne costituite da indici creati su tabelle accessibili.
ALL_IND_EXPRESSIONS	Espressioni di indici funzionali su tabelle accessibili.
ALL_IND_PARTITIONS	Tutte le partizioni dell'indice.
ALL_IND_SUBPARTITIONS	Tutte le partizioni secondarie dell'indice.
ALL_INTERNAL_TRIGGER	Descrizione di trigger interni su tabelle accessibili all'utente.
ALL_JOBS	Sinonimo per USER_JOBS.
ALL_JOIN_IND_COLUMNS	Colonne Index di join contenenti le condizioni di join.
ALL_LIBRARIES	Descrizione delle librerie accessibili all'utente.
ALL_LOBS	Descrizione dei LOB contenuti nelle tabelle accessibili all'utente.
ALL_LOB_PARTITIONS	Tutte le partizioni del LOB.
ALL_LOB_SUBPARTITIONS	Tutte le partizioni secondarie del LOB.
ALL_LOG_GROUPS	Definizioni dei gruppi di log su tabelle accessibili.
ALL_LOG_GROUP_COLUMNS	Informazioni relative alle colonne contenute nelle definizioni dei gruppi di log.
ALL_METHOD_PARAMS	Descrizione dei parametri dei metodi per i tipi accessibili all'utente.
ALL_METHOD_RESULTS	Descrizione dei risultati dei metodi per i tipi accessibili all'utente.
ALL_MVIEWS	Tutte le viste materializzate nel database.
ALL_MVIEW_AGGREGATES	Descrizione degli aggregati delle viste materializzate accessibili all'utente.
ALL_MVIEW_ANALYSIS	Descrizione delle viste materializzate accessibili all'utente.
ALL_MVIEW_DETAIL_RELATIONS	Descrizione delle tabelle relative ai dettagli delle viste materializzate accessibili all'utente.
ALL_MVIEW_JOINS	Descrizione di un join tra due colonne nella clausola WHERE di una vista materializzata accessibile all'utente.
ALL_MVIEW_KEYS	Descrizione delle colonne che compaiono nell'elenco GROUP BY di una vista materializzata accessibile all'utente.
ALL_MVIEW_LOGS	Tutti i log delle viste materializzate nel database che l'utente può visualizzare.
ALL_MVIEW_REFRESH_TIMES	Viste materializzate e relative ultime date di aggiornamento per ciascuna tabella master che l'utente può visualizzare.
ALL_NESTED_TABLES	Descrizione delle tabelle annidate nelle tabelle accessibili all'utente.
ALL_OBJECTS	Oggetti accessibili all'utente.

*(segue)*

(continua)

NOME_TABELLA	COMMENTI
ALL_OBJECT_TABLES	Descrizione di tutte le tabelle oggetto accessibili all'utente.
ALL_OPANCILLARY	Tutti gli operatori ausiliari disponibili per l'utente.
ALL_OPARGUMENTS	Tutti gli argomenti degli operatori disponibili per l'utente.
ALL_OPBINDINGS	Tutte le funzioni di binding per gli operatori disponibili per l'utente.
ALL_OPERATORS	Tutti gli operatori disponibili per l'utente.
ALL_OPERATOR_COMMENTS	Commenti per gli operatori definiti dall'utente.
ALL_OUTLINES	Sinonimo per USER_OUTLINES.
ALL_OUTLINE_HINTS	Sinonimo per USER_OUTLINE_HINTS.
ALL_PARTIAL_DROP_TABS	Tutte le tabelle con colonne parzialmente scaricate accessibili all'utente.
ALL_PART_COL_STATISTICS	Statistiche di colonna per le partizioni.
ALL_PART_HISTOGRAMS	Iistogrammi per le partizioni.
ALL_PART_INDEXES	Definizioni dell'indice di partizione.
ALL_PART_KEY_COLUMNS	Colonne della chiave di partizione.
ALL_PART_LOBS	Dettagli dei LOB di partizione.
ALL_PART_TABLES	Definizioni di tabella partizionata.
ALL_PENDING_CONV_TABLES	Tutte le tabelle accessibili all'utente che non sono aggiornate all'ultima versione del tipo.
ALL_POLICIES	Tutti i criteri per gli oggetti se l'utente dispone di privilegi di sistema oppure possiede gli oggetti.
ALL_POLICY_CONTEXTS	Tutti i gruppi di criteri definiti per tutte le tabelle o viste accessibili all'utente.
ALL_POLICY_GROUPS	Tutti i gruppi di criteri definiti per qualsiasi tabella o vista accessibili all'utente.
ALL_PROBE_OBJECTS	Tutti gli oggetti di prova.
ALL_PROCEDURES	Descrizione di tutte le procedure disponibili per l'utente.
ALL_QUEUES	Tutte le code accessibili all'utente.
ALL_QUEUE_TABLES	Tutte le tabelle di coda accessibili all'utente.
ALL_REFRESH	Tutti i gruppi di aggiornamento cui l'utente può accedere.
ALL_REFRESH_CHILDREN	Tutti gli oggetti nei gruppi di aggiornamento in cui l'utente può accedere al gruppo.
ALL_REFRESH_DEPENDENCIES	Descrizione delle tabelle di dettaglio da cui dipendono le viste materializzate per l'aggiornamento.
ALL_REFS	Descrizione di colonne REF contenute in tabelle accessibili all'utente.

(segue)

*(continua)*

NOME_TABELLA	COMMENTI
ALL_REGISTERED_MVIEWS	Viste materializzate remote di tabelle locali che l'utente può visualizzare.
ALL_REGISTERED_SNAPSHOTS	Viste materializzate remote di tabelle locali che l'utente può visualizzare.
ALL_REPAUDIT_ATTRIBUTE	Informazioni sugli attributi gestiti automaticamente per la replica.
ALL_REPAUDIT_COLUMN	Informazioni sulle colonne presenti in tutte le tabelle shadow per le tabelle replicate che sono accessibili all'utente.
ALL_REPCAT	Catalogo di replica.
ALL_REPCATLOG	Informazioni sulle richieste amministrative asincrone.
ALL_REPOCOLUMN	Colonne di livello superiore replicate (tabella) ordinate alfabeticamente in ordine crescente.
ALL_REPOCOLUMN_GROUP	Tutti i gruppi di colonne di tabelle replicate accessibili all'utente.
ALL_REPCONFICT	Tutti i conflitti con le risoluzioni disponibili per le tabelle replicate dell'utente.
ALL_REPDLL	Argomenti che non si adattano a un singolo record di log repcat.
ALL_REPOFLAVORS	Varianti definite per i gruppi di oggetti replicati.
ALL_REPOFLAVOR_COLUMNS	Colonne replicate nelle varianti.
ALL_REPOFLAVOR_OBJECTS	Oggetti replicati nelle varianti.
ALL_REPOGENERATED	Oggetti generati per supportare la replica.
ALL_REPOGENOBJECTS	Oggetti generati per supportare la replica.
ALL_REPOGROUP	Informazioni sui gruppi di oggetti replicati.
ALL_REPOGROUPED_COLUMN	Colonne in tutti i gruppi di colonne delle tabelle replicate che sono accessibili all'utente.
ALL_REPOGROUP_PRIVILEGES	Informazioni sugli utenti registrati per i privilegi sui gruppi di oggetti.
ALL_REPOKEY_COLUMNS	Colonne primarie per una tabella che usa la replica a livello di colonna.
ALL_REPOBJECT	Informazioni sugli oggetti replicati.
ALL_REPPARAMETER_COLUMN	Tutte le colonne utilizzate per risolvere i conflitti nelle tabelle replicate che sono accessibili all'utente.
ALL_REPPRIORITY	Valori e priorità corrispondenti in tutti i gruppi di priorità che sono accessibili all'utente.
ALL_REPPRIORITY_GROUP	Informazioni su tutti i gruppi di priorità che sono accessibili all'utente.
ALL_REPOPROP	Informazioni di propagazione relative agli oggetti replicati.
ALL_REPORESOLUTION	Descrizioni di tutte le risoluzioni di conflitti per le tabelle replicate che sono accessibili all'utente.
ALL_REPORESOLUTION_METHOD	Tutti i metodi di risoluzione dei conflitti accessibili all'utente.

*(segue)*

(continua)

NOME_TABELLA	COMMENTI
ALL_REPRESOLUTION_STATISTICS	Statistiche sulle risoluzioni dei conflitti per le tabelle replicate che sono accessibili all'utente.
ALL_REPRESOL_STATS_CONTROL	Informazioni sulla collection di statistiche relative alle risoluzioni di conflitti per le tabelle replicate che sono accessibili all'utente.
ALL_REPSHEMA	Informazioni sulla replica in n-modi.
ALL_REPSITES	Informazioni sulla replica in n-modi.
ALL_SECONDARY_OBJECTS	Tutti gli oggetti secondari per gli indici di dominio.
ALL_SEQUENCES	Descrizione delle sequenze accessibili all'utente.
ALL_SNAPSHOTS	Viste materializzate cui l'utente può accedere.
ALL_SNAPSHOT_LOGS	Tutti i log delle viste materializzate nel database, che l'utente può vedere.
ALL_SNAPSHOT_REFRESH_TIMES	Sinonimo di ALL_MVIEW_REFRESH_TIMES.
ALL_SOURCE	Sorgente corrente sugli oggetti memorizzati che l'utente può creare.
ALL_SOURCE_TABLES	Tabelle di origine.
ALL_SOURCE_TAB_COLUMNS	Colonne di origine.
ALL_SQLJ_TYPES	Descrizione dei tipi accessibili all'utente.
ALL_SQLJ_TYPE_ATTRS	Descrizione degli attributi dei tipi accessibili all'utente.
ALL_SQLJ_TYPE_METHODS	Descrizione dei metodi dei tipi accessibili all'utente.
ALL_STORED_SETTINGS	Impostazioni dei parametri per gli oggetti cui l'utente può accedere.
ALL_SUBPART_COL_STATISTICS	Statistiche sulle partizioni secondarie.
ALL_SUBPART_HISTOGRAMS	Iistogrammi delle partizioni secondarie.
ALL_SUBPART_KEY_COLUMNS	Chiavi delle partizioni secondarie.
ALL_SUBSCRIBED_COLUMNS	Colonne sottoscritte.
ALL_SUBSCRIBED_TABLES	Tabelle sottoscritte.
ALL_SUBSCRIPTIONS	Tutte le sottoscrizioni.
ALL_SUMDELTA	Voci di caricamento a percorso diretto accessibili all'utente.
ALL_SUMMARIES	Descrizione dei riepiloghi accessibili all'utente.
ALL_SYNONYMS	Tutti i sinonimi accessibili all'utente.
ALL_TABLES	Descrizione delle tabelle relazionali accessibili all'utente.
ALL_TAB_COLS	Colonne delle tabelle, delle viste e dei cluster dell'utente.

(segue)

*(continua)*

NOME_TABELLA	COMMENTI
ALL_TAB_COLUMNS	Colonne delle tabelle, delle viste e dei cluster dell'utente.
ALL_TAB_COL_STATISTICS	Colonne delle tabelle, delle viste e dei cluster dell'utente.
ALL_TAB_COMMENTS	Commenti sulle tabelle e le viste accessibili all'utente.
ALL_TAB_HISTOGRAMS	Istogrammi sulle colonne di tutte le tabelle visibili all'utente.
ALL_TAB_MODIFICATIONS	Informazioni sulle modifiche apportate alle tabelle.
ALL_TAB_PARTITIONS	Tutte le partizioni della tabella.
ALL_TAB_PRIVS	Concessioni sugli oggetti per i quali l'utente è il beneficiario, il concedente, il proprietario o per i quali il beneficiario è un ruolo abilitato o PUBLIC.
ALL_TAB_PRIVS_MADE	Concessioni dell'utente e sugli oggetti dell'utente.
ALL_TAB_PRIVS_REC'D	Concessioni sugli oggetti per i quali all'utente è stato concesso il ruolo PUBLIC o è stato abilitato il ruolo.
ALL_TAB_SUBPARTITIONS	Tutte le partizioni secondarie della tabella.
ALL_TRIGGER_COLS	Utilizzo delle colonne nei trigger dell'utente o nei trigger presenti sulle tabelle dell'utente.
ALL_TYPES	Descrizione dei tipi accessibili all'utente.
ALL_TYPE_ATTRS	Descrizione degli attributi dei tipi accessibili all'utente.
ALL_TYPE_METHODS	Descrizione dei metodi dei tipi accessibili all'utente.
ALL_TYPE VERSIONS	Descrizione di ciascuna versione del tipo accessibile all'utente.
ALL_UNUSED_COL_TABS	Tutte le tabelle con colonne inutilizzate accessibili all'utente.
ALL_UPDATABLE_COLUMNS	Descrizione di tutte le colonne aggiornabili.
ALL_USERS	Informazioni relative a tutti gli utenti del database.
ALL_USTATS	Tutte le statistiche.
ALL_VARRAYS	Descrizione di array variabili in tabelle accessibili all'utente.
ALL_VIEWS	Descrizione delle viste accessibili all'utente.
AUDIT_ACTIONS	Tabella descrittiva per i codici dei tipi di azione per la coda di audit. Associa i numeri dei tipi di azione ai nomi dei tipi di azioni.
CAT	Sinonimo per USER_CATALOG.
CLU	Sinonimo per USER_CLUSTERS.
COLS	Sinonimo per USER_TAB_COLUMNS.

*(segue)*

(continua)

NOME_TABELLA	COMMENTI
COLUMN_PRIVILEGES	Concessioni sulle colonne per le quali l'utente è il beneficiario, il concedente, il proprietario o per le quali il beneficiario è un ruolo abilitato o PUBLIC.
DATABASE_COMPATIBLE_LEVEL	Gruppo di parametri compatibili con il database tramite init.ora.
DBMS_ALERT_INFO	
DBMS_LOCK_ALLOCATED	
DICT	Sinonimo per DICTIONARY.
DICTIONARY	Descrizione delle tabelle e delle viste del dizionario dati.
DICT_COLUMNS	Descrizione delle colonne nelle tabelle e nelle viste del dizionario dati.
DUAL	Tabella a una riga e una colonna.
GLOBAL_NAME	Nome globale del database.
IND	Sinonimo per USER_INDEXES.
INDEX_HISTOGRAM	Informazioni statistiche sulle chiavi con conteggio di ripetizione.
INDEX_STATS	Informazioni statistiche sul b-tree.
NLS_DATABASE_PARAMETERS	Parametri NLS permanenti del database.
NLS_INSTANCE_PARAMETERS	Parametri NLS dell'istanza.
NLS_SESSION_PARAMETERS	Parametri NLS della sessione dell'utente.
OBJ	Sinonimo per USER_OBJECTS.
RESOURCE_COST	Costo per ciascuna risorsa.
ROLE_ROLE_PRIVS	Ruoli concessi ad altri ruoli.
ROLE_SYS_PRIVS	Privilegi di sistema concessi ai ruoli.
ROLE_TAB_PRIVS	Privilegi di tabella concessi ai ruoli.
SEQ	Sinonimo per USER_SEQUENCES.
SESSION_PRIVS	Privilegi che l'utente ha attualmente impostato.
SESSION_ROLES	Ruoli che risultano attualmente abilitati per l'utente.
SM\$VERSION	Sinonimo per SM_\$VERSION.
SYN	Sinonimo per USER_SYNONYMS.
TABLE_PRIVILEGES	Concessioni sugli oggetti per i quali l'utente è il beneficiario, il concedente, il proprietario o per i quali il beneficiario è un ruolo abilitato o PUBLIC.
TABS	Sinonimo per USER_TABLES.
USER_ALL_TABLES	Descrizione di tutti gli oggetti e di tutte le tabelle relazionali posseduti dall'utente.

(segue)

*(continua)*

NOME_TABELLA	COMMENTI
USER_ARGUMENTS	Argomenti in un oggetto accessibili all'utente.
USER_ASSOCIATIONS	Tutte le associazioni definite dall'utente.
USER_AUDIT_OBJECT	Record di code di audit che riguardano oggetti, in modo particolare: tabelle, cluster, viste, indici, sequenze, database link [public], sinonimi [public], procedure, trigger, segmenti di rollback, tablespace, ruoli, utenti.
USER_AUDIT_POLICIES	Tutti i criteri di audit per gli oggetti contenuti nel proprio schema.
USER_AUDIT_SESSION	Tutti i record delle code di audit che riguardano CONNECT e DISCONNECT.
USER_AUDIT_STATEMENT	Record della coda di audit che riguardano i comandi grant, revoke, audit, noaudit e alter system.
USER_AUDIT_TRAIL	Voci della coda di audit rilevanti per l'utente.
USER_BASE_TABLE_MVIEWS	Tutte le viste materializzate con log possedute dall'utente nel database.
USER_CATALOG	Tabelle, viste, sinonimi e sequenze di proprietà dell'utente.
USER_CLUSTERS	Descrizioni dei cluster dell'utente.
USER_CLUSTER_HASH_EXPRESSIONS	Funzioni di hash per i cluster hash dell'utente.
USER_CLU_COLUMNS	Corrispondenza tra le colonne della tabella e quelle del cluster.
USER_COLL_TYPES	Descrizione dei tipi di collezioni specificati di proprietà dell'utente.
USER_COL_COMMENTS	Commenti sulle colonne delle tabelle e delle viste dell'utente.
USER_COL_PRIVS	Concessioni sulle colonne per le quali l'utente è il proprietario, il concedente o il beneficiario.
USER_COL_PRIVS_MADE	Tutte le concessioni sulle colonne degli oggetti di proprietà dell'utente.
USER_COL_PRIVS_REC'D	Concessioni sulle colonne per le quali l'utente è il beneficiario.
USER_CONSTRAINTS	Definizioni dei vincoli sulle tabelle di proprietà dell'utente.
USER_CONS_COLUMNS	Informazioni relative alle colonne accessibili nelle definizioni dei vincoli.
USER_DB_LINKS	Database link di proprietà dell'utente.
USER_DEPENDENCIES	Dipendenze da e verso gli oggetti di un utente.
USER_DIMENSIONS	Descrizione degli oggetti di dimensionamento accessibili al DBA.
USER_DIM_ATTRIBUTES	Rappresentazione della relazione tra un livello di dimensionamento e una colonna dipendente dalla funzionalità.
USER_DIM_CHILD_OF	Rappresentazione di una relazione gerarchica di tipo 1:n tra una coppia di livelli in una dimensione.
USER_DIM_HIERARCHIES	Rappresentazione di una gerarchia di dimensioni.

*(segue)*

(continua)

NOME_TABELLA	COMMENTI
USER_DIM_JOIN_KEY	Rappresentazione di un join tra due tabelle di dimensionamento.
USER_DIM_LEVELS	Descrizione dei livelli di dimensionamento visibili al DBA.
USER_DIM_LEVEL_KEY	Rappresentazioni delle colonne di un livello di dimensionamento.
USER_ERRORS	Errori correnti sugli oggetti memorizzati di proprietà dell'utente.
USER_EXTENTS	Extent contenenti segmenti di proprietà dell'utente.
USER_EXTERNAL_LOCATIONS	Descrizione delle locazioni delle tabelle esterne di proprietà dell'utente.
USER_EXTERNAL_TABLES	Descrizione delle tabelle esterne di proprietà dell'utente.
USER_FREE_SPACE	Extent liberi nelle tablespace accessibili all'utente.
USER_HISTOGRAMS	Sinonimo per USER_TAB_HISTOGRAMS.
USER_INDEXES	Descrizione degli indici di proprietà dell'utente.
USER_INDEXTYPES	Tutti i tipi di indici dell'utente.
USER_INDEXTYPE_COMMENTS	Commenti per i tipi di indici definiti dall'utente.
USER_INDEXTYPE_OPERATORS	Tutti gli operatori dei tipi di indici dell'utente.
USER_IND_COLUMNS	Colonne contenenti gli indici dell'utente e gli indici sulle tabelle dell'utente.
USER_IND_EXPRESSIONS	Espressioni di indici funzionali negli indici dell'utente e negli indici sulle tabelle dell'utente.
USER_IND_PARTITIONS	Partizioni dell'indice dell'utente
USER_IND_SUBPARTITIONS	Partizioni secondarie dell'indice dell'utente.
USER_INTERNAL_TRIGGER	Descrizione dei trigger interni sulle tabelle di proprietà dell'utente.
USER_JAVA_POLICY	Permessi di sicurezza Java per l'utente corrente.
USER_JOBS	Tutti i job di proprietà di un utente.
USER_JOIN_IND_COLUMNS	Colonne Index di join che contengono le condizioni di join.
USER_LIBRARIES	Descrizione delle librerie di proprietà dell'utente.
USER_LOBS	Descrizione dei LOB di proprietà dell'utente contenuti nelle tabelle dell'utente.
USER_LOB_PARTITIONS	Partizioni del LOB dell'utente.
USER_LOB_SUBPARTITIONS	Partizioni secondarie del LOB dell'utente.
USER_LOG_GROUPS	Definizioni dei gruppi di log sulle tabelle di proprietà dell'utente.
USER_LOG_GROUP_COLUMNS	Informazioni sulle colonne contenute nelle definizioni dei gruppi di log.
USER_METHOD_PARAMS	Descrizione dei parametri dei metodi dei tipi di proprietà dell'utente.

(segue)

*(continua)*

NOME_TABELLA	COMMENTI
USER_METHOD_RESULTS	Descrizione dei risultati dei metodi dei tipi di proprietà dell'utente.
USER_MVIEWS	Tutte le viste materializzate nel database.
USER_MVIEW_AGGREGATES	Descrizione degli aggregati della vista materializzata creati dall'utente.
USER_MVIEW_ANALYSIS	Descrizione delle viste materializzate create dall'utente.
USER_MVIEW_DETAIL_RELATIONS	Descrizione delle tabelle relative ai dettagli delle viste materializzate create dall'utente.
USER_MVIEW_JOINS	Descrizione di un join tra due colonne nella clausola WHERE di una vista materializzata creata dall'utente.
USER_MVIEW_KEYS	Descrizione delle colonne che compaiono nell'elenco GROUP BY di una vista materializzata creata dall'utente.
USER_MVIEW_LOGS	Tutti i log delle viste materializzate di proprietà dell'utente.
USER_MVIEW_REFRESH_TIMES	Viste materializzate e relative ultime date di aggiornamento per ciascuna tabella master che l'utente può visualizzare.
USER_NESTED_TABLES	Descrizione delle tabelle annidate contenute nelle tabelle di proprietà dell'utente.
USER_OBJECTS	Oggetti di proprietà dell'utente.
USER_OBJECT_SIZE	Dimensioni, in byte, di vari oggetti pl/sql.
USER_OBJECT_TABLES	Descrizione delle tabelle oggetto di proprietà dell'utente.
USER_OBJ_AUDIT_OPTS	Opzioni di audit per le tabelle e le viste di proprietà dell'utente.
USER_OPANCILLARY	Tutti gli operatori ausiliari definiti dall'utente.
USER_OPARGUMENTS	Tutti gli argomenti dell'operatore per gli operatori definiti dall'utente.
USER_OPBINDINGS	Tutte le funzioni o i metodi di binding sugli operatori definiti dall'utente.
USER_OPERATORS	Tutti gli operatori dell'utente.
USER_OPERATOR_COMMENTS	Commenti per gli operatori definiti dall'utente.
USER_OUTLINES	Strutture memorizzate di proprietà dell'utente.
USER_OUTLINE_HINTS	Suggerimenti memorizzati nelle strutture di proprietà dell'utente.
USER_PARTIAL_DROP_TABS	Tabelle dell'utente con colonne inutilizzate.
USER_PART_COL_STATISTICS	Statistiche sulle colonne di partizione.
USER_PART_HISTOGRAMS	Iistogrammi sulle colonne di partizione.
USER_PART_INDEXES	Definizioni delle partizioni dell'indice.
USER_PART_KEY_COLUMNS	Chiavi di partizione.
USER_PART_LOBS	LOB di partizione.

*(segue)*

(continua)

NOME_TABELLA	COMMENTI
USER_PART_TABLES	Definizioni delle tabelle di partizione.
USER_PASSWORD_LIMITS	Visualizza i limiti della password dell'utente.
USER_PENDING_CONV_TABLES	Tutte le tabelle dell'utente che non sono aggiornate all'ultima versione del tipo.
USER_POLICIES	Tutti i criteri di sicurezza a livello di riga per le tabelle o le viste di proprietà dell'utente.
USER_POLICY_CONTEXTS	Tutti i gruppi di criteri definiti per le tabelle o le viste nello schema corrente.
USER_POLICY_GROUPS	Tutti i gruppi di criteri definiti per qualsiasi tabella o vista.
USER PROCEDURES	Descrizione delle procedure di proprietà dell'utente.
USER_PROXYES	Descrizione delle connessioni che l'utente può stabilire.
USER_QUEUES	Tutte le code di proprietà di un utente.
USER_QUEUE_SCHEDULES	I programmi di coda dell'utente.
USER_QUEUE_TABLES	Tutte le tabelle di coda create dall'utente.
USER_REFRESH	Tutti i gruppi di aggiornamento.
USER_REFRESH_CHILDREN	Tutti gli oggetti nei gruppi di aggiornamento per i quali l'utente è il proprietario del gruppo.
USER_REFS	Descrizione delle colonne REF di proprietà dell'utente contenute nelle tabelle dell'utente.
USER_REGISTERED_MVIEWS	Viste materializzate remote delle tabelle locali che attualmente utilizzano log di proprietà dell'utente.
USER_REGISTERED_SNAPSHOTS	Viste materializzate remote delle tabelle locali che attualmente utilizzano log di proprietà dell'utente.
USER_REPAUDIT_ATTRIBUTE	Informazioni sugli attributi gestiti automaticamente per la replica.
USER_REPAUDIT_COLUMN	Informazioni sulle colonne in tutte le tabelle shadow per le tabelle replicate dell'utente.
USER_REPCAT	Catalogo di replica.
USER_REPCATLOG	Informazioni sulle richieste amministrative asincrone dell'utente corrente.
USER_REPCOLUMN	Colonne replicate per la tabella dell'utente corrente in ordine crescente.
USER_REPCOLUMN_GROUP	Tutti i gruppi di colonne delle tabelle replicate dell'utente.
USER_REPCONFICT	Conflitti di replica.
USER_REPDLL	Argomenti che non rientrano in un unico record di log repcat.
USER_REPFLAVORS	Caratteristiche create dall'utente corrente per i gruppi di oggetti replicati.
USER_REPFLAVOR_COLUMNS	Colonne replicate dalle tabelle dell'utente corrente in caratteristiche.

(segue)

*(continua)*

<b>NOME_TABELLA</b>	<b>COMMENTI</b>
USER_REPFLAVOR_OBJECTS	Oggetti dell'utente replicati in varianti.
USER_REPGENERATED	Oggetti generati per l'utente corrente per supportare la replica.
USER_REPGENOBJECTS	Oggetti generati per l'utente corrente per supportare la replica.
USER_REPGROUP	Informazioni di replica relative all'utente corrente.
USER_REPGROUPED_COLUMN	Colonne in tutti i gruppi di colonne delle tabelle replicate dell'utente.
USER_REPGROUP_PRIVILEGES	Informazioni sugli utenti registrati per i privilegi sui gruppi di oggetti.
USER_REPKEY_COLUMNS	Colonne primarie per una tabella che usa la replica a livello di colonna.
USER_REPOBJECT	Informazioni di replica relative agli oggetti dell'utente corrente.
USER_REPPARAMETER_COLUMN	Tutte le colonne impiegate per risolvere i conflitti nelle tabelle replicate dell'utente.
USER_REPPRIORITY	Valori e relative priorità corrispondenti nei gruppi di priorità dell'utente.
USER_REPPRIORITY_GROUP	Informazioni sui gruppi di priorità dell'utente.
USER_REPPROP	Informazioni di propagazione sugli oggetti dell'utente corrente.
USER_REPRESOLUTION	Descrizione di tutte le risoluzioni dei conflitti per le tabelle replicate dell'utente.
USER_REPRESOLUTION_METHOD	Tutti i metodi di risoluzione dei conflitti accessibili all'utente.
USER_REPRESOLUTION_STATISTICS	Statistiche relative alle risoluzioni dei conflitti per le tabelle replicate dell'utente.
USER_REPRESOL_STATS_CONTROL	Informazioni sulla collezione di statistiche relative alle risoluzioni dei conflitti per le tabelle replicate dell'utente.
USER_REPSHEMA	Informazioni di replica in n modi sull'utente corrente.
USER_REPSITES	Informazioni di replica in n modi sull'utente corrente.
USER_RESOURCE_LIMITS	Visualizza i limiti delle risorse dell'utente.
USER_RESUMABLE	Informazioni di sessione recuperabile per l'utente corrente.
USER_ROLE_PRIVS	Ruoli concessi all'utente corrente.
USER_RSRC_CONSUMER_GROUP_PRIVS	Cambia i privilegi per i gruppi di consumer dell'utente.
USER_RSRC_MANAGER_SYSTEM_PRIVS	Privilegi di sistema per il resource manager dell'utente.
USER_RULESETS	Gruppi di regole di proprietà dell'utente.
USER_SECONDARY_OBJECTS	Tutti gli oggetti secondari per gli indici di dominio.
USER_SEGMENTS	Memoria allocata per tutti i segmenti del database.
USER_SEQUENCES	Descrizione delle sequenze di proprietà dell'utente.
USER_SNAPSHOTS	Viste materializzate che l'utente può visualizzare.

*(segue)*

(continua)

NOME_TABELLA	COMMENTI
USER_SNAPSHOT_LOGS	Tutti i log delle viste materializzate di proprietà dell'utente.
USER_SNAPSHOT_REFRESH_TIMES	Sinonimo per USER_MVIEW_REFRESH_TIMES.
USER_SOURCE	Sorgente degli oggetti memorizzati accessibili all'utente.
USER_SOURCE_TABLES	Tabelle di origine.
USER_SOURCE_TAB_COLUMNS	Colonne di origine.
USER_SQLJ_TYPES	Descrizione dei tipi di proprietà dell'utente.
USER_SQLJ_TYPE_ATTRS	Descrizione degli attributi dei tipi di proprietà dell'utente.
USER_SQLJ_TYPE_METHODS	Descrizione dei metodi dei tipi di proprietà dell'utente.
USER_STORED_SETTINGS	Impostazioni dei parametri per gli oggetti di proprietà dell'utente.
USER_SUBPART_COL_STATISTICS	Statistiche relative alle colonne di partizioni secondarie.
USER_SUBPART_HISTOGRAMS	Iistogrammi delle partizioni secondarie.
USER_SUBPART_KEY_COLUMNS	Chiavi di partizioni secondarie.
USER_SUBSCRIBED_COLUMNS	Colonne sottoscritte.
USER_SUBSCRIBED_TABLES	Tabelle sottoscritte.
USER_SUBSCRIPTIONS	Sottoscrizioni dell'utente.
USER_SUMMARIES	Descrizione dei riepiloghi creati dall'utente.
USER_SYNONYMS	Sinonimi privati dell'utente.
USER_SYS_PRIVS	Privilegi di sistema concessi all'utente corrente.
USER_TABLES	Descrizione delle tabelle relazionali di proprietà dell'utente.
USER_TABLESPACES	Descrizione delle tablespace accessibili.
USER_TAB_COLS	Colonne delle tabelle, delle viste e dei cluster dell'utente.
USER_TAB_COLUMNS	Colonne delle tabelle, delle viste e dei cluster dell'utente.
USER_TAB_COL_STATISTICS	Colonne delle tabelle, delle viste e dei cluster dell'utente.
USER_TAB_COMMENTS	Commenti sulle tabelle e sulle viste di proprietà dell'utente.
USER_TAB_HISTOGRAMS	Iistogrammi sulle colonne delle tabelle dell'utente.
USER_TAB_MODIFICATIONS	Informazioni sulle modifiche apportate alle tabelle.
USER_TAB_PARTITIONS	Partizioni delle tabelle dell'utente.
USER_TAB_PRIVS	Concessioni sugli oggetti per i quali l'utente è proprietario, concedente o beneficiario.

(segue)

*(continua)*

<b>NOME_TABELLA</b>	<b>COMMENTI</b>
USER_TAB_PRIVS_MADE	Tutti i privilegi sugli oggetti di proprietà dell'utente.
USER_TAB_PRIVS_REC'D	Concessioni sugli oggetti per i quali l'utente è beneficiario.
USER_TAB_SUBPARTITIONS	Partizioni secondarie delle tabelle dell'utente.
USER_TRANSFORMATIONS	Trasformazioni dell'utente.
USER_TRIGGERS	Trigger di proprietà dell'utente.
USER_TRIGGER_COLS	Uso delle colonne nei trigger dell'utente.
USER_TS_QUOTAS	Quote di tablespace per l'utente.
USER_TYPES	Descrizione dei tipi di proprietà dell'utente.
USER_TYPE_ATTRS	Descrizione degli attributi dei tipi di proprietà dell'utente.
USER_TYPE_METHODS	Descrizione dei metodi dei tipi di proprietà dell'utente.
USER_TYPE VERSIONS	Descrizione di ogni versione dei tipi dell'utente.
USER_UNUSED_COL_TABS	Tabelle dell'utente con colonne inutilizzate.
USER_UPDATABLE_COLUMNS	Descrizione delle colonne aggiornabili.
USER_USERS	Informazioni relative all'utente corrente.
USER_USTATS	Tutte le statistiche sulle tabelle o gli indici di proprietà dell'utente.
USER_VARRAYS	Descrizione degli array variabili contenuti nelle tabelle di proprietà dell'utente.
USER_VIEWS	Descrizione delle viste di proprietà dell'utente.
V\$ACTIVE_INSTANCES	Sinonimo per V_\$ACTIVE_INSTANCES.
V\$ACTIVE_SESS_POOL_MTH	Sinonimo per V_\$ACTIVE_SESS_POOL_MTH.
V\$BH	Sinonimo per V_\$BH.
V\$LOADCSTAT	Sinonimo per V_\$LOADCSTAT.
V\$LOADISTAT	Sinonimo per V_\$LOADISTAT.
V\$LOADPSTAT	Sinonimo per V_\$LOADPSTAT.
V\$LOADTSTAT	Sinonimo per V_\$LOADTSTAT.
V\$LOCK_ACTIVITY	Sinonimo per V_\$LOCK_ACTIVITY.
V\$MAX_ACTIVE_SESS_TARGET_MTH	Sinonimo per V_\$MAX_ACTIVE_SESS_TARGET_MTH.
V\$MLS_PARAMETERS	Sinonimo per V_\$MLS_PARAMETERS.
V\$NLS_PARAMETERS	Sinonimo per V_\$NLS_PARAMETERS.

*(segue)*

(continua)

NOME_TABELLA	COMMENTI
V\$NLS_VALID_VALUES	Sinonimo per V_\$NLS_VALID_VALUES.
V\$OPTION	Sinonimo per V_\$OPTION.
V\$PARALLEL_DEGREE_LIMIT_MTH	Sinonimo per V_\$PARALLEL_DEGREE_LIMIT_MTH.
V\$PQ_SESSTAT	Sinonimo per V_\$PQ_SESSTAT.
V\$PQ_TQSTAT	Sinonimo per V_\$PQ_TQSTAT.
V\$QUEUEING_MTH	Sinonimo per V_\$QUEUEING_MTH.
V\$RSRC_CONSUMER_GROUP	Sinonimo per V_\$RSRC_CONSUMER_GROUP.
V\$RSRC_CONSUMER_GROUP_CPU_MTH	Sinonimo per V_\$RSRC_CONSUMER_GROUP_CPU_MTH.
V\$RSRC_PLAN	Sinonimo per V_\$RSRC_PLAN.
V\$RSRC_PLAN_CPU_MTH	Sinonimo per V_\$RSRC_PLAN_CPU_MTH.
V\$SESSION_CONNECT_INFO	Sinonimo per V_\$SESSION_CONNECT_INFO.
V\$SESSION_LONGOPS	Sinonimo per V_\$SESSION_LONGOPS.
V\$TEMPORARY_LOBS	Sinonimo per V_\$TEMPORARY_LOBS.
V\$TIMEZONE_NAMES	Sinonimo per V_\$TIMEZONE_NAMES.
V\$VERSION	Sinonimo per V_\$VERSION.

## VSIZE

**VEDERE ANCHE** CARATTERI, FUNZIONI; LENGTH; NUMERI, FUNZIONI; Capitolo 8.

### SINTASSI

VSIZE(*valore*)

**DESCRIZIONE** VSIZE fornisce la dimensione di memoria di un valore in Oracle. Per le colonne di caratteri, VSIZE equivale esattamente a LENGTH. Per i numeri, è generalmente minore della lunghezza apparente, dato che i database dispongono di routine ottimizzate per la memorizzazione dei numeri.

### ESEMPI

VSIZE(12.345) = 4

## WHENEVER\*

**VEDERE ANCHE** EXECUTE, FETCH.

### SINTASSI

```
EXEC SQL WHENEVER {NOT FOUND | SQLERROR | SQL WARNING}
{CONTINUE | }
```

---

```
GOTO etichetta |
STOP |
DO routine
DO BREAK
DO CONTINUE}
```

**DESCRIZIONE** WHENEVER non è un'istruzione SQL eseguibile, bensì una direttiva del processore del linguaggio Oracle che di fatto aggiunge una condizione del tipo “IF *condizione* THEN GOTO *etichetta*” al termine di ogni istruzione SQL che segue. Mentre viene eseguita, i risultati di ogni istruzione SQL vengono controllati per vedere se soddisfano la *condizione*. In caso affermativo, il programma passa a *etichetta*.

WHENEVER può essere utilizzato con CONTINUE o con un'etichetta diversa per ciascuna delle tre possibili condizioni. Ciascuna di queste si propagherà a tutte le successive istruzioni SQL. Un nuovo comando WHENEVER con una di queste condizioni rimpiazza completamente il suo predecessore per tutte le istruzioni SQL successive.

La condizione NOT FOUND viene sollevata quando SQLCODE vale 100; questo significa, per esempio, che un FETCH non restituisce nessuna riga (ciò include anche le subquery delle istruzioni insert).

SQLERROR è vera quando SQLCODE è minore di 0. Si tratta generalmente di errori irreversibili che richiedono complesse routine di gestione.

SQLWARNING è vera quando si verifica un errore non irreversibile. In questa categoria rientrano il troncamento di stringhe di caratteri caricate all'interno di variabili host, una select list di colonne che non corrisponde all'elenco INTO delle variabili host, l'esecuzione di istruzioni di cancellazione o aggiornamento prive di clausole where e così via.

In genere, la diramazione iniziale, la continuazione, l'interruzione o l'esecuzione di routine vengono impostate all'inizio del programma, prima di qualsiasi istruzione SQL eseguibile. Comunque, all'interno di particolari blocchi logici del programma, esse possono essere sovrascritte in base alle necessità locali. Si noti che WHENEVER non sa nulla riguardo le regole di ambito dei linguaggi host, le chiamate o la diramazione. Dal momento in cui si impone un comando WHENEVER, le sue regole rimangono attive finché un altro WHENEVER (con la stessa condizione) non lo sostituisce. D'altra parte, il programma deve obbedire alle regole di ambito del linguaggio per quanto riguarda GOTO e l'etichetta.

L'opzione STOP interrompe l'esecuzione del programma; l'opzione DO chiama una routine di qualche tipo tra quelle del linguaggio host. La sintassi di questa routine dipende dal linguaggio host impiegato. DO BREAK interrompe un ciclo; DO CONTINUE esegue un'istruzione continua da un ciclo quando la condizione è soddisfatta.

Infine, all'interno di una routine di gestione degli errori, specie in una che può contenere istruzioni SQL, WHENEVER SQLERROR dovrebbe contenere CONTINUE o STOP oppure dovrebbe chiamare una routine di uscita in modo da evitare cicli infiniti nella stessa routine di gestione degli errori.

## WHENEVER OSERROR

**VEDERE ANCHE** WHENEVER SQLERROR.

### SINTASSI

WHENEVER OSERROR

```
{EXIT [SUCCESS|FAILURE|n|variabile:BindVariable] [COMMIT|ROLLBACK]
|CONTINUE [COMMIT|ROLLBACK|NONE]}
```

**DESCRIZIONE** WHENEVER OSERROR gestisce la logica di elaborazione in SQL\*PLUS nel caso in cui si verifichi un errore nel sistema operativo. EXIT induce SQL\*PLUS a uscire, mentre

CONTINUE disattiva EXIT. COMMIT comunica a SQL\*PLUS di eseguire un COMMIT prima di uscire, mentre ROLLBACK gli indica di eseguire un ROLLBACK prima di uscire. Il comportamento predefinito è NONE.

## WHENEVER SQLERROR

**VEDERE ANCHE** WHENEVER OSERROR

### SINTASSI

```
WHENEVER SQLERROR
{EXIT [SUCCESS|FAILURE|WARNING|n|variable|:BindVariable]
[COMMIT|ROLLBACK]|CONTINUE [COMMIT|ROLLBACK|NONE]}
```

**DESCRIZIONE** WHENEVER SQLERROR gestisce la logica di elaborazione in SQL\*PLUS se un comando SQL o un blocco PL/SQL incontra un errore. EXIT induce SQL\*PLUS a uscire, mentre CONTINUE disattiva EXIT. COMMIT comunica a SQL\*PLUS di eseguire un COMMIT prima di uscire, mentre ROLLBACK gli indica di eseguire un ROLLBACK prima di uscire. Il comportamento predefinito è NONE.

**NOTA** *Gli errori di SQL e di PL/SQL inizializzano l'elaborazione di WHENEVER SQLERROR. Gli errori di SQL\*Plus invece non hanno lo stesso effetto.*

È possibile porre WHENEVER SQLERROR in un file di avvio davanti a un numero qualsiasi di istruzioni SQL. Ogni WHENEVER SQLERROR sovrascrive quello precedente e rimane attivo fino a che non ne interviene uno successivo a sovrascriverlo.

### ESEMPIO

Nell'esempio seguente, CREATE TABLE non riesce perché il letterale KEENE non è racchiuso tra apici, quindi i successivi comandi update e select non verranno mai eseguiti. WHENEVER SQLERROR esce immediatamente da SQL\*PLUS e passa il codice sql.sqlcode all'host:

```
WHENEVER SQLERROR EXIT SQL.SQLCODE;
```

```
create table KEENE as
select * from COMFORT
where Citta = KEENE;

update KEENE set Mezzogiorno = 35;

select * from KEENE;
```

## WHERE

**VEDERE ANCHE** DELETE, OPERATORI LOGICI, PRECEDENZE, SELECT, OPERATORI SINTATTICI, UPDATE.

### SINTASSI

```
DELETE FROM [utente.]tabella ...
[ WHERE condizione ]
SELECT ...
[ WHERE condizione ] UPDATE [utente.]tabella [alias] ...
[WHERE condizione ]
```

**DESCRIZIONE** WHERE definisce le condizioni logiche che vincolano la selezione delle righe da restituire, cancellare o aggiornare.

Una condizione può essere definita in uno dei modi seguenti:

- Un confronto tra espressioni ( $=$ ,  $>$ ,  $\neq$  e così via).
- Un confronto tra un'espressione e una query.
- Un confronto tra un elenco di espressioni e un elenco di espressioni derivanti da una query.
- Un confronto tra un'espressione e alcuni o tutti (ANY o ALL) i membri di un elenco o tra un'espressione e i valori restituiti da una query.
- Un controllo che un'espressione sia o non sia (IN o NOT IN) in un elenco o tra i risultati di una query.
- Una verifica che qualcosa sia contenuta o meno (BETWEEN o NOT BETWEEN) tra due valori.
- Una verifica che un'espressione sia NULL o meno (IS NULL o IS NOT NULL).
- Una verifica di esistenza o meno (EXISTS o NOT EXISTS) dei risultati di una query.
- Una combinazione delle condizioni precedenti correlate mediante gli operatori logici AND e OR.

## ESEMPI

```
where Section ANY ('A','C','D');
where Citta !=ALL (select Citta from DISLOCAZIONE);
where Sezione IN ('A','C','D');
where Citta NOT IN (select Citta from DISLOCAZIONE);
where Pagina BETWEEN 4 and 7;
where Capacita IS NULL;
where EXISTS (select * from LAVORATORE where Eta  90);
where Sezione = 'A' or Sezione = 'B' and Pagina = 1;
```

## WHILE (PL/SQL)

Vedere LOOP, Capitolo 27.

## WIDTH\_BUCKET

**VEDERE ANCHE** NTILE.

### SINTASSI

```
WIDTH_BUCKET ( expr , valore_min , valore_max , num_bucket )
```

**DESCRIZIONE** WIDTH\_BUCKET costruisce istogrammi di pari larghezza, in cui l'intervallo dell'istogramma è suddiviso in intervalli che hanno una dimensione identica (Si confronti questa funzione con NTILE, che crea istogrammi di pari altezza). Idealmente, ogni bucket è un intervallo “chiuso-aperto” della linea del numero reale. Per una determinata espressione, WIDTH\_BUCKET restituisce il numero di bucket in cui rientrerà il valore di questa espressione dopo essere stata valutata. *expr* è l'espressione (un numero o un valore data/ora) per la quale viene creato l'istogramma. Se *expr* vale NULL, l'espressione restituisce NULL. *valore\_min* e *valore\_max* sono due espressioni che si risolvono nelle estremità dell'intervallo accettabile di *expr*. Entrambe le espressioni devono valere un numero o valori di data/ora, ma nessuna può valere NULL. *num\_bucket* è un'espressione che si risolve in una costante che indica il numero di bucket. Questa espressione deve valere un numero intero positivo.

## WRAP (SQL\*PLUS)

Vedere SET.

## WRAPPING

Wrapping significa spezzare la parte finale di un'intestazione, di un titolo o di una colonna portandola su un'altra linea quando viene superato il margine di visualizzazione. È l'effetto opposto di TRUNCATE. Vedere COLUMN.

## XML

XML (eXtensible Markup Language) è una sintassi standardizzata per descrivere dati gerarchici. XML fornisce un formato universale per documenti e dati strutturati. La sua sintassi basata su tag gli consente di gestire con molta flessibilità dati complessi. HTML comunica a un browser Web come presentare i dati, mentre XML indica alle applicazioni il significato dei dati. Per degli esempi, si rimanda al Capitolo 41.

## XMLType

XMLType è un tipo di dati disponibile a partire da Oracle9i che serve per memorizzare ed eseguire query sui dati XML nel database. In qualità di tipo, XMLType dispone di funzioni membro per accedere, estrarre ed eseguire query sui dati XML con una classe di operazioni note con il nome di espressioni XPath. I package SYS\_XMLGEN, SYS\_XMLAGG e DBMS\_XMLGEN creano valori XMLType a partire da dati relazionali a oggetti già esistenti. Quando si stabilisce che una colonna utilizzerà il tipo di dati XMLType, Oracle memorizza internamente i dati in un tipo di dati CLOB.

Il listato seguente mostra la creazione di una tabella che utilizza il tipo di dati XMLType:

datatype:

```
create table MIA_TABELLA_XML
(Chiavel      NUMBER,
Colonna_Xml   SYS.XMLTYPE);

insert into MIA_TABELLA_XML (Chiavel, Colonna_Xml)
values (1, SYS.XMLTYPE.CREATEXML
('<book>
<title>Complete Reference</title>
<chapter num= "42">
<title>Ending</title>
<text>This is the end of the book.</text>
</chapter>
</book>'));
```

È possibile recuperare i dati dalla colonna XMLType tramite il metodo GETCLOBVAL:

```
select M.Xml_Column.GETCLOBVAL() as XML_Data
from MIA_TABELLA_XML
where Chiavel = 1;
```

Per ulteriori dettagli su XMLType e i relativi metodi correlati, si rimanda al Capitolo 41.

# Indice analitico

## A

- ' (apice singolo) 798
- " (doppi apici) 794
- ! (punto esclamativo) 794
- # (cancelletto) 795
- \$ (dollaro) 795
- % (segno percentuale) 795
- %FOUND 796
- %ISOPEN 796
- %ROWCOUNT 797
- %ROWTYPE 797
- %TYPE 798
- & o && 798
- ( ) (parentesi) 799
- \* (moltiplicazione) 799
- \*\* (esponenziale) 799
- + (addizione) 800
- (sottrazione [forma 1]) 800
- (trattino [forma 2]) 800
- (commento) 800
- . (punto [forma 1]) 801
- . (punto [forma 2]) 801
- / (barra [forma 2]) 802
- / (divisione [forma 1]) 802
- /\* \*/ (commento) 802
- : (due punti, prefisso di variabile host) 803
- := (uguale a) 803
- ; (punto e virgola) 803
- <<>> (delimitatore di nome dell'etichetta PL/SQL) 804
- ? (punto interrogativo) 795
- @ (chiocciola [forma 1]) 804
- @ (chiocciola [forma 2]) 804
- @@ 804
- \_ (trattino di sottolineatura) 794
- { } (parentesi graffe) 804
- | (barra verticale) 805
- || (concatenazione) 805
- ABS, funzione 141–142, 806
- ACCEPT 806
- accesso concorrente 807
- ACOS, funzione 146, 807
- ADD\_MONTHS, funzione 162, 807
- addizione, funzione 139–140
- alias 807–808
  - nelle viste 201
  - per colonne 200–201
- ALL 152–153, 808
- ALLOCATE 809
- alter cluster 809–810
- alter database 810–818
- alter dimension 818–820
- alter function 820
- alter index 820–824
- alter indextype 824–825
- alter java 825
- alter materialized view 825–830
- alter materialized view log 830–831
- alter outline 831–832
- alter package 832
- alter procedure 832
- alter profile 832–833
- alter resource cost 834
- alter role 834
- alter rollback segment 834–835
- alter sequence 835–836
- alter session 836–837
- alter system 837–839
- alter table 839–857
- alter tablespace 858–859
- alter trigger 859–860
- alter type 860–862
- alter user 862–864
- alter view 864
- ambienti, modifica 17–18
- analisi procedurale 29
- ANALYZE 865–867
- AND-EQUAL 679
- ANSI 867
- ANY 867–868
- Apache 719–722, 735–736
- Apache JServ 725
- APPEND
  - funzione 557–558, 868
  - suggerimento 264
- append, comando 101
- ARCH, processo 869
- ARCHIVE LOG 869
- area di contesto 869
- aritmetiche, funzioni 139–140
- array, elaborazione 869
- array variabile 870
- array variabili
  - creazione 525–526
  - definizione 70
  - descrizione 526–527
  - inserimento di record 527–528
  - problemi di gestione 535–537
  - selezione di dati 528–531
- AS 870
- ASCII 870
  - funzione 129
- ASCIISTR 871
- ASIN, funzione 146, 871
- ASSOCIATE STATISTICS 871–872
- ATAN, funzione 146, 873
- ATAN2, funzione 146, 873
- ATTRIBUTE 873
- AUDIT
  - comando 874–876
  - operazione 877
- AUTOCOMMIT 877

autocommit 264–265  
AVG, funzione 149, 877

**B**

B-TREE 883  
backup  
  Export 758–764  
  fuori linea 767–768, 878  
  Import 764–767  
  in linea 768–770  
  n linea 877  
  Oracle Enterprise Manager 771–772  
  Recovery Manager 771, 1077  
BEGIN 878  
BFILE 878  
BFILENAME 878  
BIN\_TO\_NUM 879  
BITAND 879  
bitmap, indici 353–354  
BLOB 879  
BREAK 882  
break on, comando 94  
break on Row, comando 243  
btitle, comando 92, 243,  
  882–883  
buffer  
  database 884  
  redo log 884  
  SQL 99  
  SQLPLUS 883  
Business Intelligence Services 727  
Business Logic Services 724–725

**C**

cache 884  
  di dizionario 985  
Caching Services 730–733  
CALL 884  
carattere jolly 52  
CASE 887–888  
  funzione 298–300  
  istruzioni PL/SQL 465–466  
CAST 888  
CATSEARCH 428, 888  
CEIL, funzione 142, 888–889  
CHAINED\_ROWS 665  
CHANGE 889  
change, comando 100

CHARTOROWID 890  
checkpoint 890  
chiamata ricorsiva 890  
chiave  
  candidata 305  
  composta 890  
  di cluster 893–894  
  esterna 23, 37, 307–308,  
    890  
  intelligente 38–39  
  primaria 21, 307, 890  
  sovraffllo 39  
  unica 891  
CHR, funzione 129, 891  
cicli  
  a cursore 892  
  DO-WHILE, Java 588  
  e cursori Java 589  
  FOR 891  
  FOR, Java 588–589  
  PL/SQL FOR 461–462  
  PL/SQL FOR a cursore  
    462–463  
  PL/SQL semplici 458–459  
  PL/SQL semplici a cursore  
    459–461  
  PL/SQL WHILE 463–465  
  semplici 891  
  WHILE 891  
  WHILE, Java 588  
classi  
  accesso 615–619  
  caricamento nel database  
    614–615  
  Java 591–595  
  JDBC 599–605  
  SQLJ 607–609  
clausole  
  connect by 229–233  
  group by 195–199  
  having 195–199  
  INTERSECT 219–221  
  MINUS 219–221  
  on commit preserve rows 225  
  order by 129–133, 197–198  
  start with 237  
  UNION 219–221  
  when, clausola 267–271  
  where 129–133  
CLEAR 892–893  
clear breaks, comando 107  
clear columns, comando 107  
clear computes, comando 107  
Clickstream 733  
CLOSE 893  
cluster 361–362, 893  
chiave 893  
hash 894  
indice 894  
COALESCE, funzione 154, 894  
coda di audit 894  
coerenza di lettura 373  
colonne  
  aggiunta 311–312  
  chiavi intelligenti 38  
  cluster, dizionario dati 643  
  di join 895  
  dizionario dati 628–630  
  eliminazione 313  
  indicizzate 355  
  indicizzate, dizionario dati  
    641  
  integrità referenziale 1025  
  intestazioni con SQLPLUS  
    98–99  
  larghezza 302  
  LONG-LOB 312  
  modifica 311–312  
  nomi in lingua corrente  
    23–25  
  oggetti 563–564  
  ordine in break on 243  
  pseudocolonna User  
    347, 387–389  
  rinominare con alias 200–201  
  unione 205–206  
  uso di order by 205  
vincoli 895  
vincolo, dizionario dati 635–636  
column, comando 93–94,  
  895–898  
comandi  
  append 101  
  break on 94, 240–243  
  break on Row 243  
  btitle 92, 243  
  change 100  
  clear breaks 107  
  clear columns 107  
  clear computes 107  
  column 93–94  
  commit 264–265  
  compute avg 94–95  
  compute sum 240–243  
  connect 337–340  
  copy 389–390  
  create function 492–499  
  create materialized view  
    396–402  
  create materialized view log  
    410–413

- create package 499–502  
 create procedure 490–492  
 create synonym 340  
 create tablespace 358–359  
 delete 271–272  
 eseguibili Java 583–591  
 eseguibili, PL/SQL 455–466  
 explain plan 706–709  
 fold\_after 258–259  
 fold\_before 258–259  
 grant 335  
 host 104  
 insert 261–264  
 merge 274–276  
 remark 91  
 rollback 264–265  
 save 102  
 set autotrace on 702–706  
 set headsep 92  
 set linesize 95  
 set newpage 96  
 set pagesize 95  
 set pause 102  
 set termout off 252  
 set termout on 252  
 show all 257  
 shutdown 741  
 spool 96–98, 257  
 SQLLDR 367–371  
 SQLPLUS, aggiunta 105  
 SQLPLUS di base 88  
 start 105  
 store 103  
 ttitle 92, 243  
 update 272–274  
**COMMENT** 898  
**COMMIT** 264–265  
 PL/SQL 899  
 SQL 898–899  
**Communication Services**  
 719–722  
**COMPARE**, funzione 554–555  
 compiti dell'applicazione  
     documentazione 31–32  
     profilo 30–32  
 compiti dell'applicazione  
     comprensione 28–30  
**COMPOSE** 899–900  
**COMPUTE** 900–901  
 compute avg, comando 94–95  
 concatenazione, funzione  
     112–113  
**CONNECT**  
     forma 1 901  
     forma 2 (SQL) 901–902  
**CONNECT BY** 902–903  
 connect by, clausola 229–233  
 connect, comando 337–340  
**CONNECT**, ruolo 334  
 constructor method 527  
**CONTAINS** 427, 903  
**Content Management Services**  
     722–723  
 control file 366–367, 904  
 conversione  
     automatica dei tipi di dati  
         187–190  
     funzioni elementari 185–190  
     funzioni specializzate 190  
**CONVERT** 905–906  
 copia e incollamento complessi  
     281–285  
 copy, comando 389–390  
**COPY**, funzione 560, 906–907  
**CORR** 907  
**COS**, funzione 146, 908  
**COSH**, funzione 146, 908  
**COUNT**, funzione 149, 908  
**COVAR\_POP** 908  
**COVAR\_SAMP** 908–909  
 create cluster 909–910  
 create context 910–911  
 create controlfile 911–912  
 create database 912–914  
 create database link 914–915  
 create dimension 916–917  
 create directory 917  
 create function, comando  
     492–499  
 create global temporary table 225  
 create index 919–923  
 create indextype 923–924  
 create java 924–925  
 create library 925  
 create materialized view 925–931  
 create materialized view,  
     comando 396–402  
 create materialized view log  
     931–932  
 create materialized view log,  
     comando 410–413  
 create operator 932–933  
 create outline 933–934  
 create package body, comando  
     934–935  
 create package, comando  
     499–502, 934  
 create pfile 935  
 create procedure 935–936  
 create procedure, comando  
     490–492  
 create profile 936–938  
 create rollback segment 938  
 create role 938  
 create schema 939  
 create sequence 939–940  
 create spfile 940  
 create synonym 940–941  
 create synonym 340  
 create table 941–953  
 create tablespace 358–359,  
     953–955  
 create temporary tablespace 955  
 create trigger 956–957  
 create type 957–961  
 create type body 962–963  
 create user 963–964  
 create view 964–966  
**CTXCAT** 966  
**CUBE**, funzione 225–229, 966  
**CUME\_DIST** 966–967  
**CURRENT\_DATE** 967  
**CURRENT\_DATE**, funzione 160  
**CURRENT\_TIMESTAMP** 967  
**CURRVAL** 967

**D**

- database**  
 accesso a dati esterni  
     431–432  
 accesso al pubblico 348–349  
 accesso alle classi 615–619  
 accesso concorrente 807  
 aggiunta di testo 415–416  
 allocazione e gestione dello  
     spazio per gli oggetti  
     743–754  
 aree di memoria 742–743  
 array variabili 525–531  
 attivazione e disattivazione  
     di ruoli 345–346  
 avvio e arresto 740–741  
 buffer 884  
 cache 884  
 calcoli dello spazio 751–754  
 caricamento delle classi  
     614–615  
 checkpoint 890  
 chiave primaria 21  
 chiavi esterne 23, 37  
 chiavi intelligenti 38  
 chiuso 969  
 cluster 361–362  
 collegamenti dinamici  
     389–390  
 colonne indicizzate 355  
 concessione di privilegi  
     a ruoli 342–343

- concessione di risorse limitate 349  
concessione di ruoli a ruoli 343  
concessione di un ruolo a utenti 343–344  
connessioni esplicite e predefinite 384  
control file 904  
create synonym 340  
creazione 739–740  
creazione di indici 352  
creazione di ruoli 342  
creazione di segmenti di rollback 754–757  
creazione di un utente 329–330  
creazione di viste 63–66  
deadlock 976  
distribuito 969  
e tecnologie Web 81–84  
eliminazione di ruoli 346  
entità 996  
extent liberi 750–751  
fattore umano 27  
file dei parametri di inizializzazione 743  
gestione delle password 330–334  
gestione di segmenti di rollback 754–757  
implicazioni della storage clause 744–747  
IMS 18  
indici basati su indici 357  
inserimento dei dati 35  
integrità a livello dei nomi 36–37  
integrità referenziale 1025  
limiti al riutilizzo delle password 333–334  
link, funzionamento 379–380  
modalità ARCHIVELOG 375  
modalità NOARCHIVELOG 375  
modello entità-relazioni 996  
navigazione tra i dati 22–23  
nome 969  
nomi in lingua corrente 23–25  
normalizzazione 16, 20  
oggetti 578  
oggetto di 970  
posizionamento degli indici 355–356  
privilegi di oggetti 329  
privilegi di sistema 329  
query e report 35–36  
query remote 380–381  
record logici e fisici 370–371  
relazionale 10–11, 67  
relazionale a oggetti 4, 67  
relazionale, rischi 16–17  
remoto, connessione 390–391  
replica 1082  
revoca di privilegi 335  
revoca di privilegi da un ruolo 346  
ricostruzione degli indici 356–357  
ridimensionamento degli oggetti 751–755  
savepoint 1090–1091  
scadenza delle password 331–333  
segmenti di indice 747–748  
segmenti di rollback 360, 748–749  
segmenti di tabella 747  
segmenti di undo 748–749  
segmenti temporanei 749–750  
sequenze 362–363  
SGA 742  
shared pool 743  
spazio libero 750  
standard di denominazione 18–19  
stringa di connessione 384–385  
tabelle annidate 531–535  
tabelle esterne, creazione 432–440  
tabelle esterne, limitazioni 440–442  
tabelle esterne, usi possibili 440–442  
tabelle esterne, vantaggi 440–442  
trasmissione di privilegi 341  
trigger a livello di 471  
trigger di eventi 483–484  
VSAM 18
- database link  
aggiornamenti remoti 382–383  
condivisi 385–386  
definizione 969  
 dizionario dati 646–647  
 funzionamento 379–380  
 pubblici e privati 383–384
- query remote 380–381  
sintassi 383–386  
stringa di connessione 384–385  
uso per sinonimi 381–382  
uso per viste 381–382  
Database Services 734–735  
datazione giuliana 970  
date  
 ADD\_MONTHS 162  
 aggiunta di mesi 162  
 aritmetica 159–168  
 clausola where 178–179  
 combinazione di funzioni 167–168  
 CURRENT\_DATE 160  
 datazione giuliana 970  
 DBTIMEZONE 163  
 differenza tra due 160–161  
 EXTRACT 163, 180  
 formati 171–172, 970–972  
 FROM\_TZ 163  
 funzioni 163–164, 973–974  
 fusi orari 163  
 gestione di più secoli 179–180  
 GREATEST 162–164  
 LAST\_DAY 163, 166  
 LEAST 162–164  
 MONTHS\_BETWEEN 167  
 NEW\_TIME 163, 175–176  
 NEXT\_DAY 164  
 NUMTODSINTERVAL 163  
 ROUND 163, 168  
 sottrazione di mesi 162  
 SYS\_EXTRACT\_UTC 163  
 SYSDATE 160  
 SYSTIMESTAMP 160, 164  
 TO\_CHAR 169–178  
 TO\_DATE 169–178  
 TO\_DSINTERVAL 164  
 TO\_TIMESTAMP 164  
 TO\_TIMESTAMP\_TZ 164  
 TO\_YMINTERVAL 164  
 TRUNC 164  
 TZ\_OFFSET 164
- dati  
 a lunghezza variabile, caricamento 366–367  
 aggiornamento 272–274  
 arrotondamento 304–305  
 avvio del caricamento 367–371  
 caricamento a percorso diretto 376–377  
 chiave primaria 21

- comando insert 261–264  
 comprensione 32–36  
 delete 271–272  
 duplicati, conservazione (trigger) 476–477  
 eliminazione 271–272  
 esterni 431–432  
 file 974  
 gestione delle operazioni di caricamento 373  
 indipendenza 974  
 inscindibili, modelli 34  
 inserimenti in più tabelle 267–271  
 inserimento 35, 261–264  
 inserimento di un'ora 262  
 insert con select 263–264  
 maiuscole e minuscole 26–27  
 manipolazione con JDBC 602–605  
 manipolazione con SQLJ 609–612  
 manipolazione mediante viste oggetto 517  
 messa a punto delle operazioni di caricamento 375–377  
 modello commerciale 34  
 navigazione 22–23  
 normalizzazione 20, 21  
**NUMBER**, precisione 303–304  
 parole in lingua corrente 26  
 query remote 380–381  
 ripetizione delle operazioni di caricamento 374  
 salvataggio nelle query flashback 445–446  
 scelta delle denominazioni 20  
 selezione con SQL 44–47  
 selezione da array variabili 528–531  
 update con NULL 274  
 update con select 273–274
- DBA**  
 allocazione e gestione dello spazio per gli oggetti 743–754  
 avvio e arresto del database 740–741  
 creazione di un database 739–740  
 gestione delle aree di memoria 742–743  
 regole per i calcoli dello spazio 751–754
- ridimensionamento degli oggetti 751–755  
 ruolo 334–335  
 uso di Oracle Enterprise Manager 740  
**DBMS\_LOB**, package 547–561  
**DBMS\_MVIEW** 411  
**DBMS OLAP** 412  
**DBMS\_OUTPUT**, package 495–496, 548  
**DBTIMEZONE**, funzione 163, 976  
**DBWR** 976  
**DCL** 968  
**DDL** 968–969  
 deadlock 976  
**DECLARE** 977  
**DECLARE CURSOR** 977  
**DECLARE DATABASE** 978  
**DECLARE STATEMENT** 978  
**DECLARE TABLE** 978  
**DECODE**  
 funzione 289–292  
 Greater Than e Less Than 296–298  
 in DECODE 293–296  
 sostituzione di valori 292–293  
**DECODE**, funzione 191–193, 282, 979  
**DECOMPOSE** 979  
**DEL** 980  
 delete, comando 271–272, 980–982  
 delimitatori, SQLPLUS 100  
 denominazione  
 problemi comuni 25  
 denominazione standard 18–19  
**DENSE\_RANK** 982  
**DEREF** 982  
**DEREF**, funzione 567  
**DESCRIBE** 983  
 diagrammi a barre e grafici 278–280  
**DISASSOCIATE STATISTICS** 984  
**DISCONNECT** 984  
**distinct** 984  
**DISTINCT**, opzione 152–153  
 distribuite 1074  
 divisione, funzione 139–140  
 dizionario dati  
 audit 662–663  
 blocchi 985  
 blocchi sulle definizioni 880  
 cache 985  
 catalogo 626  
 cluster 642–643  
 colonne 628–630  
 colonne cluster 643  
 colonne indicizzate 641  
 colonne vincolo 635–636  
 commenti di colonna 638–639  
 commenti di tabella 637–638  
 database link 646–647  
**DICT\_COLUMNS** 624–625  
**DICTIONARY (DICT)** 624–625  
 dimensione del codice 652  
 dimensioni 652–653  
 dizionario dati 650  
 eccezioni ai vincoli 636–637  
 errori di codice 651–652  
 indici 639–640  
 interdipendenze 666  
 librerie 667  
 limiti delle risorse 660  
**LOB** 646  
 log delle viste materializzate 649  
 nomenclatura 624  
 oggetti 626  
 Oracle Label Security 666  
 partizioni e partizioni secondarie 656–659  
 privilegi di colonna 660–661  
 privilegi di sistema 661  
 privilegi di tabella 660  
 procedure, funzioni e package 650–652  
 quote di spazio 654–655  
 ruoli 661–662  
 segmenti ed extent 655–656  
 sequenze 633–634  
 sinonimi 632–633  
 spazio libero 659  
 tabelle 626–628  
 tabelle di prestazioni  
 dinamiche V\$ 663–668  
 tablespace 653–654  
**USER\_CATALOG (CAT)** 626  
 utenti 659  
 vincoli 634  
 viste 1153–1167  
 viste materializzate 647–649  
 viste NLS 667
- DML** 969  
**DML LOCK** 985  
**DO WHILE**, cicli Java 588

Document Type Definition 773  
 DROP CLUSTER 985–986  
 DROP CONTEXT 986  
 DROP DATABASE LINK 986  
 DROP DIMENSION 986  
 DROP DIRECTORY 987  
 DROP FUNCTION 987  
 DROP INDEX 987  
 DROP INDEXTYPE 987  
 DROP JAVA 988  
 DROP LIBRARY 988  
 DROP MATERIALIZED VIEW 988  
 DROP MATERIALIZED VIEW LOG 988  
 DROP OPERATOR 989  
 DROP OUTLINE 989  
 DROP PACKAGE 989  
 DROP PROCEDURE 989  
 DROP PROFILE 990  
 DROP ROLE 990  
 DROP ROLLBACK SEGMENT 990  
 DROP SEQUENCE 990  
 DROP SNAPSHOT 991  
 DROP SNAPSHOT LOG 991  
 DROP SYNONYM 991  
 DROP TABLE 991  
 DROP TABLESPACE 991  
 DROP TRIGGER 992  
 DROP TYPE 992  
 DROP TYPE BODY 992  
 DROP USER 992  
 DROP VIEW 993  
 DROPJAVA 993  
 DUAL, tabella 161, 993  
 DUMP 993–994

**E**

EBCDIC 994  
 EDIT 994  
 editing, SQLPLUS 103  
 editor  
   a linea di comando 99–102  
   definizione con login.sql 104  
 EMPTY\_BLOB 995  
 EMPTY\_CLOB 995  
 END 995  
 enqueue 996  
 entità 996  
 equi-join 996  
 ERASE, funzione 558–559  
 esclusione di individui e rami 233–235

eventi  
   DDL 480  
   del database 483–484  
   di sistema, attributi 481–483  
 EXCEPTION 997–1000  
 EXCEPTION\_INIT 1000  
 EXCEPTIONS 636–637  
 EXECUTE 1000–1001  
 EXECUTE IMMEDIATE 1001  
 EXISTS 211–212, 1002  
 EXISTSNODE 1002  
 EXIT 1002–1003  
 EXP 1003  
 EXP, funzione 143–144  
 EXPLAIN PLAN 1003  
 explain plan 706–709  
 Export  
   esportazione delle tablespace 762–763  
   esportazione di partizioni 764  
   esportazioni coerenti 759–762  
   opzioni runtime 760–761  
   utility 758–764, 1004  
 extent  
   iniziale 358  
   liberi 750–751  
 EXTRACT, funzione 163, 180, 1004–1005

**F**

fattore umano 27  
 FETCH 1005–1006  
 FIRST 1008  
 FIRST\_VALUE 1008  
 flashback, query 443–448  
 FLOOR 1008–1009  
 FLOOR, funzione 142  
 fold\_after, comando 258–259  
 fold\_before, comando 258–259  
 FOR  
   cicli 891  
   cicli a cursore PL/SQL 462–463  
   cicli Java 588–589  
   cicli PL/SQL 461–462  
 formattazione  
   avanzata dei report 239–252  
   di numeri 255–256  
   TO\_CHAR e TO\_DATE 169–178  
   ttitle e btitle 243  
 FROM 1009  
 FROM\_TZ, funzione 163, 1009–1010

FULL OUTER JOIN 1010  
 FULL TABLE SCAN 1010  
 funzioni  
   a valore singolo e di gruppo, combinazione 149–151  
   ABS 141–142  
   ACOS 146  
   ADD\_MONTHS 162  
   addizione 139–140  
   APPEND 557–558  
   aritmetiche 139–140  
   ASCII 129  
   ASIN 146  
   ATAN 146  
   ATAN2 146  
   AVG 149  
   CASE 298–300  
   CEIL 142  
   CHR 129  
   COALESCE 154  
   combinazione 116–118  
   COMPARE 554–555  
   concatenazione 112–113  
   conversione 904–905  
   conversione automatica di tipi di dati 187–189  
   COPY 560  
   CORR 908  
   COSH 146  
   COUNT 149  
   CUBE 225–229  
   date 163–164  
   DBTIMEZONE 163  
   DECODE 191–193, 282, 289–292  
   DEREF 567  
   di conversione elementari 185–190  
   di conversione specializzate 190  
   di data, combinazione 167–168  
   di elenco 153–154, 1010  
   di gruppo 146–153, 1010–1011  
   di gruppo, opzione ALL 152–153  
   di gruppo, uso di DISTINCT 152–153  
   di regressione 1012  
   di stringa con order by 129–133  
   di stringa con where 129–133  
   divisione 139–140  
   e procedure PL/SQL 490  
   ERASE 558–559  
   EXP 143–144

EXTRACT 163, 180  
 FLOOR 142  
 FROM\_TZ 163  
 GETLENGTH 553–554  
 GREATEST 154, 162–164  
 GROUPING 225–229  
 in order by 277  
 indici basati su 357  
 INITCAP 119–121  
 INSTR 124–129, 552–553  
 INTERSECT 685–688  
 LAST\_DAY 163, 166  
 LEAST 154, 162–164  
 LENGTH 121  
 LN 143–144  
 LOG 143–144  
 LOWER 119–121  
 LPAD 114–115  
 LTRIM 115–116  
 MAX 149, 154  
 MIN 149, 154  
 MINUS 685–688  
 MOD 142–143  
 moltiplicazione 139–140  
 MONTHS\_BETWEEN 163, 167  
 NEW\_TIME 163, 175–176  
 NEXT\_DAY 163, 164  
 notazione 110–112  
 numeriche 1046–1047  
 numeriche, classi 135–136  
 NUMTODSINTERVAL 163  
 NUMTOYMINTEGERVAL, 163  
 NVL 140–141  
 per caratteri 885–887  
 per date 973–974  
 per stringhe 110–112  
 per valori singoli 139–146  
 personalizzate, creazione 496–498  
 PL/SQL, compilazione 503–504  
 PL/SQL, denominazione 499  
 PL/SQL, eliminazione 504–505  
 PL/SQL, sostituzione 504  
 POWER 143  
 precedenza 156–157  
 radice quadrata 143  
 RANK 224  
 REF 567  
 REPLACE 285–287  
 ricerca di righe 154–155  
 ROLLUP 225–229  
 ROUND 144–145, 163, 168  
 RPAD 114–115  
 RTRIM 115–116

SESSIONTIMEZONE 163  
 SIGN 145–146  
 SIN 146  
 SINH 146  
 sostituzione di valori NULL 140–141  
 sottrazione 139–140  
 SOUNDEX 131–132  
 SQRT 143  
 STDDEV 151  
 stringa 111  
 SUBSTR 121–124, 551–552  
 SUM 149  
 SYS\_EXTRACT\_UTC 163  
 SYSDATE 160  
 SYSTIMESTAMP 160, 164  
 TABLE 534–535  
 TAN 146  
 TANH 146  
 TO\_CHAR 169–178, 185  
 TO\_DATE 169–178, 185  
 TO\_DSINTERVAL 164  
 TO\_NUMBER 185  
 TO\_TIMESTAMP 164  
 TO\_TIMESTAMP\_TZ 164  
 TO\_YMINTERVAL 164  
 TRANSLATE 190–191, 280–281  
 trasformazione 184–185, 190–193  
 trigonometriche 146  
 TRIM 115–116, 118–119, 559  
 TRUNC 144–145, 164  
 TZ\_OFFSET 164  
 UNION 685–688  
 UPPER 119–121  
 uso delle parentesi 156–157  
 valore assoluto 141–142  
 VALUE 570–571  
 VARIANCE 151  
 WRITE 556  
 fusi orari 163

GREATEST, funzione 154, 162–164  
 GROUP BY 1015–1016  
 group by, clausola 195–199  
 GROUP\_ID 1016  
 GROUPING, funzione 225–229, 1016  
 GROUPING\_ID 1016  
 gruppi  
 funzioni 146–153  
 funzioni e NULL 146–148  
 indici 429–430  
 raggruppamenti complessi 223–225  
 viste 199–201

## H

HASH JOIN 696–697  
 having  
 clausola 195–199  
 logica 203–205  
 nelle viste 203–205  
 HELP 1017  
 HEXTORAW 1017  
 HOST 1017  
 host, comando 104

## I

IDEPTREE 666  
 IF 1018  
 if, then, else 289–292  
 IIS 722  
 Import 764–767  
 importazione in account differenti 766–767  
 parametri 765–766  
 requisiti dei segmenti di undo 764–767  
 utility 1018

IMS 18

IN 1018–1019  
 INDEX RANGE SCAN 674  
 INDEX UNIQUE SCAN 674

indici 679  
 basati su funzioni 357  
 bitmap 353–354, 879  
 cluster 894  
 compressi 1019  
 concatenati 1019  
 condizioni di equivalenza 677–678  
 creazione 352

## G

gestione delle eccezioni 466–468  
 GET 1012  
 GETLENGTH, funzione 553–554  
 glogin.sql 104  
 GOTO 1012–1013  
 GRANT 1013–1014  
 grant, comando 335  
 GREATEST 1014–1015

- CTXCAT 966  
definizione 351  
dizionario dati 639–640  
globali 1019  
gruppi 429–430  
locali 1019  
mess a punto 681  
operazioni che usano 673–681  
posizionamento 355–356  
Reverse 1020  
ricostruzione 356–357  
segmenti 747–748  
segmenti di 1020  
selettivi 678  
sincronizzazione 428  
testuali 1020  
unici 1020  
unicità 353  
utilità 354–355
- Information Management System. *Vedi* *IMS*
- init.ora 1020
- INITCAP, funzione 119–121, 1020
- INPUT 1021  
insert, comando 261–264, 1021–1022
- INSTR 1024  
INSTR, funzione 124–129, 552–553
- Inter-ORB Protocol 83
- INTERSECT  
clausola 219–221  
funzione 685–688, 1025
- INTERVAL DAY TO SECOND 1025
- INTERVAL YEAR TO MONTH 1025
- IS NULL 1025–1026
- iSQL\*Plus 735
- istogramma 1026
- J**
- Java  
accesso alle classi 615–619  
Apache JServ 725  
cicli FOR 588–589  
classi 591–595  
classi JDBC 599–605  
comandi eseguibili 583–591  
componenti business per 724  
Cursori e cicli 589  
dichiarazioni 582–583
- e PL/SQL 581–582  
e tecnologie Web 83–84  
e XSU 782–783  
gestione delle eccezioni 590  
kit Java Messaging Service 729  
loadjava 616  
logica condizionale 584–587  
OracleJSP 725–726  
parole riservate 591  
SQLJ 605–607  
stored procedure 613–619  
tipi di dati 583
- JDBC  
classi 599–605  
manipolazione dei dati 602–605
- join  
colonne di 895  
comando 1027  
di più di due tabelle 692–693  
equi-join 996  
FULL OUTER JOIN 1010  
HASH JOIN 696–697  
interno 218, 1027  
MERGE JOIN 693–694  
naturale 218, 1027  
non-equì-join 1044  
outer join 212–218, 697  
sostituzione di NOT IN  
con outer join 215–217
- K**
- kernel 1028
- L**
- Label Security 350  
LAG 1028  
large object. *Vedi* *LOB*  
LAST 1028  
LAST\_DAY, funzione 163, 166, 1029  
LAST\_VALUE 1029  
LDAP 730  
LEAD 1029  
LEAST, funzione 154, 162–164, 1029–1030  
LENGTH, funzione 121, 1030  
LEVEL 1030  
LGWR 1030  
LIKE 1030–1031  
LIKE, operatore 52–54
- LIST 1031  
LN, funzione 143–144, 1031  
loadjava 616, 1032
- LOB 70–71  
aggiornamento dei valori 545  
BLOB 879  
cancellazione 561  
CLOB 893  
funzione APPEND 557–558  
funzione COMPARE 554–555  
funzione COPY 560  
funzione ERASE 558–559  
funzione GETLENGTH 553–554  
funzione INSTR 552–553  
funzione SUBSTR 551–552  
funzione TRIM 559  
funzione WRITE 556  
inizializzazione dei valori 543–544  
inserimento con subquery 545  
NCLOB 1037  
package DBMS\_LOB 547–561  
parametri di memorizzazione 541–543  
procedura READ 547–551  
tipi di dati 539–540  
USER\_LOBS 646  
uso delle funzioni di stringa 546–547  
uso di funzioni e procedure per BFILE 561
- locale 969  
LOCALTIMESTAMP 1032  
LOCK TABLE 1032–1033  
log delle viste materializzate 400  
LOG, funzione 143–144  
login.sql 104  
LOWER, funzione 119–121, 1034  
LPAD, funzione 114–115, 1034  
LTRIM, funzione 115–116, 1035
- M**
- maiuscole e minuscole 26–27  
MAKE\_REF 1035  
marcatore di posizione 52  
mask.sql 256–257  
MAX, funzione 149, 154, 1035  
merge, comando 274–276  
MERGE JOIN 693–694

- metodi  
constructor method 527  
creazione 520–522  
gestione 522–523  
MIN, funzione 149, 154,  
1035–1036  
MINUS  
clausola 219–221  
funzione 685–688, 1036  
MOD, funzione 142–143,  
1036–1037  
modello  
commerciale 34  
commerciale inscindibile 34  
entità-relazioni 996  
moltiplicazione, funzioni  
139–140  
MONTHS\_BETWEEN 1037  
MONTHS\_BETWEEN, funzione  
163, 167  
MOUNT 741
- N**
- National Language Support 132  
NCHR 1037  
NEAR 1038  
NESTED LOOPS 694–696  
NEW\_TIME, funzione  
163, 175–176  
NEXT\_DAY, funzione 163, 164  
NLS\_CHARSET\_DECL\_LEN  
1038  
NLS\_CHARSET\_ID 1038–1039  
NLS\_CHARSET\_NAME 1039  
NLS\_INITCAP 1039  
NLS\_LOWER 1039  
NLS\_UPPER 1039–1040  
NLSSORT 1040  
NOAUDIT 1040–1042  
nodo  
foglia 230  
principale 230  
NOLOGGING 1042  
nomi  
di servizio 384  
in lingua corrente 23  
integrità a livello di 36–37  
maiuscole e minuscole 26–27  
normalizzazione 27, 36–38  
singolari 37–38  
sinonimi 38  
non-equ-join 1044  
normalizzazione 16, 20  
dei dati 21
- modello logico 20–21  
nomi 27, 36–38  
NOT 1044  
NOT EXISTS 1044  
NOT IN  
sostituzione  
con NOT EXISTS 217–218  
sostituzione con outer join  
215–217  
notazione esadecimale 996  
NTILE 1045  
NULL 140, 1044–1045  
nelle funzioni di gruppo  
146–148  
sostituzione di valori  
140–141  
update con 274  
valore 53–54  
NULLIF 1045  
NUMTODSINTERVAL 1048  
NUMTODSINTERVAL, funzione  
163  
NUMTOYMINTERVAL 1048  
NUMTOYMINTERVAL,  
funzione 163  
NUMWIDTH 1048  
NVL, funzione 140–141,  
1048–1049  
NVL2 1049
- O**
- oggetti  
array variabili 70  
comuni 72–78  
convenzioni  
di denominazione 71–72  
LOB 70–71  
riferimenti 71  
struttura 73–75  
tabelle annidate 70  
tipi di dati astratti 69, 76–78  
uso 67–68  
viste 71  
OID 1049  
valori 573–574  
on commit preserve rows 225  
OPEN 1049–1050  
OPEN CURSOR 1050  
operatori  
insiemistici 1051  
LIKE 52–54  
logici 1051–1052  
precedenza 1063  
relazionali 1051
- sintattici 1052–1053  
OPSS LOGIN 1053  
OR 1053  
Oracle  
area di contesto 869  
aritmetica delle date 159–168  
classi di funzioni numeriche  
135–136  
combinazione di funzioni  
116–118  
controllo dell'ambiente  
SQLPLUS 105–107  
creazione di tabelle 301–309  
creazione di un utente  
329–330  
database link 379–386  
funzioni stringa 111  
gestione delle password  
330–334  
join di più di due tabelle  
692–693  
kernel 1028  
Label Security 350  
limiti al riutilizzo  
delle password 333–334  
ottimizzatore 669–671, 1055  
PL/SQL 451–468  
Real Application Cluster  
1054  
Recovery Manager 771  
revoca di privilegi 335  
sintassi di Oracle9i  
per gli outer join 214–215  
SQL\*Loader 365–378  
stringhe 110  
supporto alle lingue nazionali  
132–133  
tabella DUAL 161  
tipi di dati 109  
uso degli oggetti 67–68  
Virtual Private Database 349  
XML 773–787  
Oracle Advanced Security 734  
Oracle Database Cache 730–733  
Oracle Discoverer 4i Viewer 727  
Oracle Enterprise Manager  
734, 740, 771–772, 1053  
Oracle Forms Services 724–725  
Oracle Internet File System 722  
Oracle Label Security 666  
Oracle Net 384  
Oracle Portal 84, 728  
Oracle Reports Services 727  
Oracle SQLJ 729  
Oracle Text 109  
ABOUT, operatore 428  
aggiunta di testo al database

- 415–416  
 caratteri jolly 424–425  
 corrispondenza esatta  
   di più parole 419–422  
 corrispondenza esatta  
   di una frase 422–423  
 corrispondenza esatta  
   di una parola 418–419  
 corrispondenze non esatte  
   425–426  
 operatori principali  
   di CONTAINS 427  
 parole con la stessa radice  
   425  
 parole dal suono simile  
   426–428  
 query di testo 417–418  
 ricerche di parole vicine  
   423–424
- Oracle Universal Installer 739  
 Oracle Web Cache 732–733  
**Oracle9iAS**  
 Apache 719–722, 735–736  
 Apache JServ 725  
 Business Intelligence Services 727  
 Business Logic Services 724–725  
 Caching Services 730–733  
 Clickstream 733  
 Communication Services 719–722  
 componenti business per Java 724  
 Content Management Services 722–723  
 Database Services 734–735  
 Discoverer 727  
 gestione del file system 723  
 IIS 722  
 interprete Perl 727  
 iSQL\*Plus 735  
 Java Messaging Service 729  
 LDAP 730  
 Oracle Advanced Security 734  
 Oracle Database Cache 730–733  
 Oracle Discoverer 4i Viewer 727  
 Oracle Enterprise Manager 734  
 Oracle Forms Services 724–725  
 Oracle Internet File System 722–723  
 Oracle Portal 728  
 Oracle Reports Services 727
- Oracle SQLJ 729  
 Oracle Web Cache 732–733  
 OracleJSP 725–726  
 PL/SQL Server Pages (PSP) 726  
 Portal Services 727–729  
 Portal-to-Go 728  
 Presentation Services 725–727  
 sviluppo XML 730  
 System Services 733–734  
   Ultra Search 723–724
- OracleJSP 725–726  
**ORDBMS** 6, 67  
 order by  
   clausola 197–198  
   con colonne e funzioni  
     di gruppo 205  
     funzioni in 277  
     nelle viste 203, 315
- ottimizzatore  
   accesso alle tabelle 671–673  
   AND-EQUAL di più indici 679  
   condizioni di equivalenza 677–678  
   explain plan 706–709  
   filtro di righe 709–710  
   HASH JOIN 696–697  
   INLIST ITERATION  
     di più scansioni 680  
   INTERSECT 685–688  
   join di più di due tabelle 692–693  
   MERGE JOIN 693–694  
   messa a punto degli indici 681  
   MINUS 685–688  
   NESTED LOOPS 694–696  
   operazioni che usano  
     gli indici 673–681  
   Oracle 1055  
   ordinamento di righe 682–683  
   outer join 697  
   percorso di esecuzione 702–709  
   query che utilizzano cluster 712–713  
   query che utilizzano database link 712  
   query che utilizzano sequenze 711–712  
   query con clausole connect by 710–711  
   raggruppamento di righe 683–684  
   selezione dalle viste 689–690  
   selezione di righe
- per l'aggiornamento 688–689  
 selezioni da subquery 690–691  
 set autotrace on 702–706  
 strutture memorizzate 707–709  
**TABLE ACCESS BY ROWID** 672–673  
**TABLE ACCESS FULL** 671–672  
**UNION** 685–688  
 uso di RowNum 684–685  
 outer join 212–218
- P**
- package  
   compilazione 503–504  
   DBMS\_LOB 547–561  
   DBMS\_OUTPUT 495, 548  
   definizione 1056  
   e procedure 490  
   eliminazione 504–505  
   inizializzazione 501–502  
   sostituzione 504
- Palo Alto Research Center 17  
 parentesi, uso 156–157  
**PASSWORD** 1059–1060  
 password  
   aggiunta a un ruolo 344  
   gestione 330–334  
   limiti al riutilizzo 333–334  
   rimozione da un ruolo 345  
   scadenza 331–333
- PAUSE 1060  
**PCTFREE** 1060  
**PCTUSED** 1061  
**PERCENT\_RANK** 1061  
**PERCENTILE\_CONT** 1061–1062  
**PERCENTILE\_DISC** 1062  
**PL/SQL**  
   a oggetti 576–577  
   cicli FOR a cursore 462–463  
   cicli FOR 461–462  
   cicli semplici 458–459  
   cicli semplici a cursore 459–461  
   cicli WHILE 463–465  
   codice sorgente di oggetti 502–503  
   comandi eseguibili 455–466  
   **COMMIT** 899  
   create function 492–499  
   create package 499–502  
   create procedure 490–492

- creazione di funzioni personalizzate 496–498 cursore 968 denominazione di funzioni 499 denominazione di procedure 499 dichiarazione di variabili 1152 e Java 581–582 esecuzione delle procedure 488–489 gestione delle eccezioni 466–468 initializzazione dei package 501–502 istruzioni CASE 465–466 logica condizionale 456–458 nozioni di base 451–452 package DBMS\_OUTPUT 495–496 personalizzazione di condizioni di errore 498–499 PL/SQL Server Pages (PSP) 726 procedure e funzioni 490 procedure e package 490 ricerca degli errori nelle procedure 495 riferimenti a tabelle remote 494–495 sezione delle dichiarazioni 452–455 tipi di dati 454 tipi di trigger 470–471 PLAN\_TABLE 665–666 PMON 1072 Portal Services 727–729 Portal-to-Go 728 POWER, funzione 143, 1062 pragma 1062 precedenza 156–157 precompilatore 1064 PREPARE 1064 Presentation Services 725–727 PRIMARY KEY, vincolo 307 PRINT 1064 privilegi accesso al pubblico 348–349 concessione di risorse limitate 349 di colonna, dizionario dati 660–661 di oggetti del database 329 di oggetto, revoca 346–347 di sistema 329 di sistema, dizionario dati 661 di tabella, dizionario dati 660 elenco dei 1065–1071 non concessi 340 per viste materializzate 394 revoca 335 revoca da un ruolo 346 trasmissione 341 UPDATE a colonne specifiche 346 procedure e funzioni PL/SQL 490 e package PL/SQL 490 PL/SQL, compilazione 503–504 PL/SQL, denominazione 499 PL/SQL, eliminazione 504–505 PL/SQL, esecuzione 488–489 PL/SQL, sostituzione 504 ricerca degli errori 495 riferimenti a tabelle remote 494–495 PRODUCT\_USER\_PROFILE 1072 progettazione dieci comandamenti 40 orientata agli oggetti 78–79 PROMPT 1072
- Q**
- query ad albero 1073 che utilizzano cluster 712–713 che utilizzano database link 712 che utilizzano sequenze 711–712 clausola connect by 229–233 clausola INTERSECT 219–221 clausola MINUS 219–221 clausola UNION 219–221 coerenza di lettura 373 con clausole connect by 710–711 coordinazione di test logici 209–210 correlate 1073 di testo 417–418 di testo, corrispondenza esatta di una parola 418–419 e report 35–36
- esclusione di individui e rami 233–235 fase di definizione 980 flashback 443–448, 1074 flashback basate su SCN 447–448 flashback basate sull'ora 444–445 flashback, salvare i dati 445–446 join interni 218 join naturali 218 ordine di esecuzione 198–199 outer join 212–218 padre 1074 percorso di esecuzione 401–402, 702–709 personalizzazione del processo 783–784 principale 1074 raggruppamenti complessi 223–225 start with 237 su tabelle annidate 533–534 subquery avanzate 207–212 subquery correlate 208–209 subquery IN 222 sulle viste oggetto 575–576 tabelle temporanee 225 QUIT 1074 quote di spazio 654–655
- R**
- radice quadrata 143 RAISE 1075 RANK, funzione 224 RATIO\_TO\_REPORT 1075 RAWTOHEX 1076 RAWTONHEX 1076 RDBMS 6, 67 READ, procedura 547–551 Real Application Cluster 1054 RECORD 1076 RECOVER 1076–1077 Recovery Manager 771, 1077 REF, funzione 567 REFTOHEX 1078 REMARK 1079 remark, comando 91 remoto 970 RENAME 1079–1080 REPFOOTER 1080 REPHEADER 1080–1081

REPLACE, funzione 285–287,  
1081

report  
aggiunta di linee vuote 257  
aggiunta di viste 244  
break on Row 243  
comando break on 240–243  
comando compute sum  
240–243  
creazione con SQLPLUS  
89–99  
e query 35–36  
formattazione avanzata  
239–252  
formattazione con ttitle e  
btitle 243  
formattazione di numeri  
255–256  
mask.sql 256–257  
ordine delle colonne  
in break on 243  
processo di creazione 89  
variabili SQLPLUS 252

RESOURCE, ruolo 334

REVOKE 1082–1084

riferimenti 71

righe

- blocco a livello di 880
- filtro 709–710
- funzioni di ricerca 154–155
- in catena 889
- inserimento in tabelle oggetto 565
- intestazione 1025
- logiche e fisiche 370–371
- oggetti 563–564
- ordinamento 682–683
- raggruppamento 195–206,  
683–684
- selezione per l'aggiornamento  
688–689

ROLL FORWARD 1084

rollback 264–265

- comando 1084–1085
- segmenti 360, 748–749
- segmenti, assegnamento  
di transazioni 757
- segmenti, attivazione 755
- segmenti, creazione 754–757
- segmenti, dimensione  
massima 756
- segmento differito 983

ROLLUP, funzione 225–229,  
1085

ROUND, funzione 144–  
145, 163, 168, 1085–1086

ROW\_NUMBER 1087  
ROWID 1087  
ROWIDTOCHAR 1087  
ROWIDTONCHAR 1087  
ROWNUM 1088  
RowNum 684–685  
RPAD, funzione 114–115, 1088  
RTRIM, funzione 115–116, 1088  
RUN 1089  
ruoli  
aggiunta di una password 344  
attivazione e disattivazione  
345–346  
concessione a ruoli 343  
concessione di privilegi  
342–343  
CONNECT 334  
creazione 342  
DBA 334–335  
dizionario dati 661–662  
eliminazione 346  
RESOURCE 334  
revoca di privilegi 346  
rimozione di una password  
345

## S

SAVE 1089

savepoint 265–266, 1090–1091

SCORE 1091

segmenti

- di indice 747–748
- di rollback 360, 748–749
- di rollback, assegnamento  
di transazioni 757
- di rollback, creazione  
754–757
- di rollback differiti 983
- di rollback, dimensione  
massima 756
- di rollback, gestione 754–757
- di tabella 358, 747
- di undo 748–749
- ed extent 655–656
- temporanei 749–750

segmenti ed extent 655–656

SELECT 1092–1097

SELECT...INTO 1098

sequenze 362–363

SESSIONTIMEZONE 1099

SESSIONTIMEZONE, funzione

163

SET 1099–1103

set autotrace on 702–706

SET CONDITION 1103  
SET CONSTRAINTS 1103–1104

set headsep, comando 92  
set linesize, comando 95  
set newpage, comando 96  
set pagesize, comando 95  
set pause, comando 102  
SET ROLE 1104  
set termout off, comando 252  
set termout on, comando 252

SET TRANSACTION  
1104–1105  
SGA 742  
shared pool 743  
SHOW 1105–1106  
show all, comando 257  
shutdown, comando 741  
SIGN, funzione 145–146, 1106  
SIN, funzione 146, 1106  
SINH, funzione 146, 1107  
sinonimi  
dizionario dati 632–633  
per la trasparenza  
di dislocazione 386–387  
pubblici 1073  
uso di database link 381–382

sottrazione, funzione 139–140

SOUNDEX, funzione 131–132,  
1107–1108

SPOOL 1108

spool, comando 96–98, 257  
SQL

- area di contesto 869
- autocommit 264–265
- buffer 99
- carattere jolly 52
- CASE 298–300
- clausole 892
- COMMIT, comando 898–899
- commit 264–265
- commit implicito 266
- CONNECT 901–902
- creazione di tabelle 301–309
- CURSOR 968
- DECODE 289–292
- delete 271–272
- e SQLPLUS, distinzione 92
- e tecnologie Web 82–83
- eliminazione di caratteri 280
- Greater Than e Less Than  
in DECODE 296–298
- if, then, else 289–292
- inserimenti in più tabelle  
267–271
- inserimento di un'ora 262
- insert 261–264

- linguaggio 8  
 logica della combinazione 56  
 marcatore di posizione 52  
 merge 274–276  
 operatore LIKE 52–54  
 parole riservate 1057–1059  
 rollback 264–265  
 rollback automatico 267  
 savepoint 265–266  
 select, from, where e order by 47–49  
 selezione dei dati da tabelle con 44–47  
 sostituzione di valori 292–293  
 stile 42–43  
 terminatore 47  
 test logici su un elenco di valori 55  
 test logici su un singolo valore 50  
 TRANSLATE 280–281  
 update 272–274  
 update con NULL 274  
 update con select 273–274  
 uso delle subquery con where 57–61  
 uso di APPEND 264  
 uso di insert con select 263–264  
 verifiche di valori singoli 49–52  
 verifiche semplici con un elenco di valori 54–56  
 when 267–271
- SQL\*Loader**  
 avvio del caricamento 367–371  
 caricamento a percorso diretto 376–377  
 caricamento di dati a lunghezza variabile 366–367  
 control file 366–367  
 gestione delle operazioni di caricamento 373  
 messa a punto delle operazioni di caricamento 375–377  
 opzioni 368  
 record logici e fisici 370–371  
 ripetizione delle operazioni di caricamento 374  
 sintassi del control file 371–373
- SQL\*PLUS** 41  
**SQLCODE** 1109  
**SQLERRM** 1109
- SQL**  
 classi 607–609  
 configurazione 606–607  
 introduzione 605–607  
 manipolazione dei dati 609–612  
 Oracle9iAS 729  
**SQLLDR**, comando 367–371, 1110–1116  
**SQLPLUS**  
 aggiunta di comandi 105  
 aggiunta di linee vuote ai report 257  
 aggiunta di viste 244  
 append 101  
 AUTOCOMMIT 877  
 break on 94, 240–243  
 break on Row 243  
 btitle 92  
 buffer 883  
 buffer SQL 99  
 change 100  
 clear breaks 107  
 clear columns 107  
 clear computes 107  
 collegamenti dinamici 389  
 column 93–94  
 comandi di base 88  
 compute avg 94–95  
 compute sum 240–243  
 controllo dell’ambiente 105–107  
 copia e incollamento complessi 281–285  
 copy 389–390  
 creazione di report 89–99  
 delimitatori 100  
 diagrammi a barre e grafici 278–280  
 e SQL, distinzione 92  
 editing 103  
 editor a linea di comando 99–102  
 fold\_after 258–259  
 fold\_before 258–259  
 formattazione avanzata 239–252  
 formattazione di numeri 255–256  
 glogin.sql 104  
 host 104  
 intestazioni di colonna 98–99  
 login.sql 104  
 mask.sql 256–257  
 ordine delle colonne in break on 243  
 remark 91
- set headsep 92  
 set linesize 95  
 set newpage 96  
 set pagesize 95  
 set pause 102  
 set termout off 252  
 set termout on 252  
 show all 257  
 sintassi 1116–1117  
 spool 96–98, 257  
 start 105  
 store 103  
 title, comando 92  
 variabili 252–255  
**SQRT**, funzione 143  
 start, comando 105  
 start with, clausola 237  
**STDDEV**, funzione 151, 1118  
**STDDEV\_POP** 1119  
**STDDEV\_SAMP** 1119  
 stile dei comandi SQL 42–43  
**STORAGE** 1119–1121  
 storage clause 744–747  
**STORE** 1121  
 store, comando 103  
 stored procedure Java 613–619  
 stringhe 110  
 occorrenze in stringhe più lunghe 285–287  
 strutture memorizzate 707–709  
 subquery  
 avanzate 207–212  
 con where 57–61  
 coordinazione di test logici 209–210  
 correlate 208–209  
 elenchi di valori 59–61  
 IN 222  
 inserimento di LOB con 545  
 outer join 212–218  
 selezioni da 690–691  
 uso di EXISTS 211–212  
 valori singoli 58–59  
**SUBSTR**, funzione 121–124, 551–552  
 suggerimenti APPEND 264  
**SUM**, funzione 149  
 supporto alle lingue nazionali 132–133  
**SYS\_CONNECT\_BY\_PATH** 1123  
**SYS\_CONTEXT** 1123  
**SYS\_DBURIGEN** 1123–1124  
**SYS\_EXTRACT\_UTC** 1124  
**SYS\_EXTRACT\_UTC**, funzione 163

SYS\_GUID 1124  
 SYS\_TYPEID 1124  
 SYS\_XMLAGG 1125  
 SYS\_XMGEN 1125  
 SYSDATE, funzione 160, 1125  
 System Global Area. *Vedi SGA*  
 System Services 733–734  
 SYSTEM, tablespace 358  
 SYSTIMESTAMP, funzione 160, 164

**T**

tabelle  
 accesso (ottimizzatore) 671–673  
 aggiunta di colonne 311–312  
 alias 808  
 annidate 70, 531–535  
 annidate, inserimento di record 532–533  
 annidate, problemi di gestione 535–537  
 annidate, query 533–534  
 arrotondamento dei dati 304–305  
 assegnazione di nomi ai vincoli 308–309  
 blocco a livello di riga 880  
 caratteristiche principali 6  
 chiave candidata 305  
 chiave esterna 23, 307–308  
 chiave primaria 307  
 chiavi intelligenti 38  
 cluster 361–362  
 colonne indicizzate 355  
 combinazione 61–63  
 creazione 301–309  
 creazione di viste 314–316  
 da un'altra tabella 316–318  
 di informazioni 7–8  
 di prestazioni dinamiche V\$ 663–668  
 di solo indice 318–319  
 dizionario dati 626–628  
 DUAL 161, 993  
 eliminazione 309  
 esterne, creazione 432–440  
 esterne, limitazioni 440–442  
 esterne, opzioni 435–440  
 esterne, usi possibili 440–442  
 esterne, vantaggi 440–442  
 FULL TABLE SCAN 1010  
 indici 351–357  
 indici basati su indici 357

inserimenti in diverse 267–271  
 larghezza delle colonne 302  
 modifica 309–313  
 modifica di colonne 311–312  
 nomi in lingua corrente 23–25  
 oggetto, aggiornamenti 566  
 oggetto, cancellazioni 566  
 oggetto, funzione REF 567  
 oggetto, funzione DEREF 567  
 oggetto, funzione VALUE 570–571  
 oggetto, inserimento di righe 565  
 oggetto, selezione di valori 565–566  
 oggetto, riferimenti non validi 571  
 partizionate 319–324  
 precisione per i numeri 303–304  
 raggruppamenti complessi 223–225  
 record logici e fisici 370–371  
 remote, riferimenti 494–495  
 ricostruzione degli indici 356–357  
 ridefinizione in linea 324–327  
 rinominare le colonne con alias 200–201  
 scelta delle denominazioni 20  
 segmenti 358, 747  
 segmenti di indice 747–748  
 selezione dei dati con SQL 44–47  
 sequenze 362–363  
 stabilità delle viste 314–315  
 tablespace di indici 307  
 tablespace temporanee 359–360  
 temporanee 225  
 vincoli di controllo 308  
 vincoli in create table 305  
 vincoli UNIQUE 305  
 viste di sola lettura 316  
 viste materializzate 393–414  
 viste oggetto con REF 571–576  
 tabelle partizionate  
 creazione 319–322  
 gestione 323  
 indicizzazione delle partizioni 322–323  
 partizionamento a elenco 322  
 partizioni secondarie 321–322  
 TABLE ACCESS BY ROWID 672–673  
 TABLE ACCESS FULL 671–672  
 TABLE, funzione 534–535  
 tablespace  
 allocazione dello spazio 744  
 create tablespace 358–359  
 di indici 307  
 di undo 360–361, 749, 756–757  
 dizionario dati 653–654  
 esportazione 762–763  
 extent liberi 750–751  
 storage clause 744–747  
 SYSTEM 358  
 temporanee 359–360  
 trasportabili 1128–1129  
 TAN, funzione 146, 1130  
 TANH, funzione 146, 1130  
 terminatore SQL 47  
 test logici  
 su un elenco di valori 55  
 su un singolo valore 50  
 TIMESTAMP, tipo di dati 181–182  
 TIMING 1131–1132  
 tipi di dati 109  
 astratti 69, 509–514, 872  
 astratti, selezione 76–78  
 astratti USER\_TYPES 643  
 attributi 644  
 BFILE 878  
 CHAR 890  
 CLOB 893  
 constructor method 527  
 conversione automatica 187–189  
 Java 583  
 LOB 70–71, 539–540  
 LONG RAW 1034  
 metodi 520–522, 644–645  
 NCHAR 1037  
 NCLOB 1037  
 NUMBER 1045  
 NVARCHAR2 1048  
 PL/SQL 454  
 principali 974–975  
 RAW 1075  
 TIMESTAMP 181–182, 1130  
 TIMESTAMP WITH LOCAL TIME ZONE 1131  
 TIMESTAMP WITH TIME ZONE 1131  
 TO\_CHAR, funzione 169–178, 185, 1132

- TO\_CLOB, funzione 1133  
 TO\_DATE, funzione 169–178,  
   185, 1133  
 TO\_DSINTERVAL, funzione  
   164, 1133  
 TO\_LOB, funzione 1134  
 TO\_MULTI\_BYTE, funzione  
   1134  
 TO\_NCHAR, funzione 1134  
 TO\_NCLOB, funzione 1135  
 TO\_NUMBER, funzione  
   185, 1135  
 TO\_SINGLE\_BYTE, funzione  
   1135  
 TO\_TIMESTAMP, funzione  
   164, 1136  
 TO\_TIMESTAMP\_TZ, funzione  
   164, 1136  
 TO\_YMINTERVAL, funzione  
   164, 1136  
 transazioni  
   assegnamento a segmenti  
 di rollback 757  
   coerenza di lettura 895  
   commit 264–265  
   commit implicito 266  
   elaborazione 994  
   rollback 264–265  
   rollback automatico 267  
   savepoint 265–266  
**TRANSLATE** 1137  
**TRANSLATE ... USING** 1137  
**TRANSLATE**, funzione  
   190–191, 280–281  
 trasformazione, funzioni 184–185  
**TREAT** 1137–1138  
 trigger  
   a livello di database 471  
   a livello di istruzione 470  
   a livello di riga 470  
   assegnazione dei nomi 479  
   attivazione e disattivazione  
 484–485  
   BEFORE e AFTER 471  
   chiamata di procedure 479  
   condizioni di errore 477–479  
   conservazione  
 di dati duplicati 476–477  
   di eventi DDL 480  
   di eventi del database  
 483–484  
   di schema 471  
   dizionario dati 650  
   DML, combinazione  
 474–475  
   eliminazione 485  
   impostazione di valori inseriti  
     475–476  
     INSTEAD OF 471, 518–520  
     sintassi 472–484  
     sostituzione 485  
   TRIM, funzione 115–116,  
    118–119, 559, 1138  
   TRUNC, funzione 144–145, 164,  
    168, 1139  
   ttitle, comando 92, 243, 1140  
   tupla 7  
   TYPE 1141  
   TZ\_OFFSET, funzione  
    164, 1141
- U**
- UID 1141  
   Ultra Search 723–724  
   UNDEFINE 1142  
   UNDERLINE 1142  
   undo  
     conservazione 748  
     segmenti 748–749  
     tablespace 749, 756–757  
   undo, tablespace 360–361  
   UNION, clausola 219–221  
   UNION, funzione 685–688,  
    1142  
   unione di colonne 205–206  
   UNISTR 1142  
   UPDATE 1143–1144  
   update, comando 272–274  
   UPPER, funzione 119–121, 1146  
   UROWID 1146  
   USER 1147  
     User, pseudocolonna 347  
     USER\_CATALOG (CAT) 626  
     USER\_CLU\_COLUMNS 643  
     USER\_CLUSTERS (CLU) 642  
     USER\_COL\_PRIVS 660–661  
     USER\_CONS\_COLUMNS  
 635–636  
     USER\_CONSTRAINTS 634  
     USER\_DB\_LINKS 646–647  
     USER\_DEPENDENCIES 666  
     USER\_ERRORS 651–652  
     USER\_EXTENTS 655–656  
     USER\_IND\_COLUMNS 641  
     USER\_INDEXES (IND)  
 639–640  
     USER\_MVIEW\_LOGS 649  
     USER\_MVIEWS 648  
     USER\_OBJECT\_SIZE 652  
     USER\_OBJECTS (OBJ) 626  
     USER\_RESOURCE\_LIMITS  
       660
- USER\_SEGMENTS 655–656  
 USER\_SEQUENCES (SEQ)  
   633–634  
 USER\_SOURCE 650–652  
 USER\_SYNONYMS (SYN)  
   632–633  
 USER\_SYS\_PRIVS 661  
 USER\_TAB\_COLUMNS (COLS)  
   628  
 USER\_TAB\_COMMENTS  
   637–638  
 USER\_TAB\_PRIVS 660  
 USER\_TABLES (TABS)  
   626–634  
 USER\_TABLESPACES 653–654  
 USER\_TRIGGERS 650  
 USER\_TS\_QUOTAS 654–655  
 USER\_USERS 659  
 USER\_VIEWS 630–632  
 USERENV 1147  
 USERNAME 1147  
 utenti  
   concessione di ruoli a  
    343–344  
   concessioni possibili  
    336–349  
   creazione 329–330  
   dizionario dati 659  
   gestione delle password  
    330–334  
   passaggio ad altri con connect  
    337–340  
   scadenza delle password  
    331–333  
   trasmissione di privilegi 341
- V**
- valori  
   inseriti, impostazione  
     (trigger) 475–476  
   LOB, aggiornamento 545  
   LOB, inizializzazione  
    543–544  
   LOB, uso di DBMS\_LOB  
    547–561  
   NULL 53–54, 140  
   OID 573–574  
   selezione dalle tabelle oggetto  
    565–566  
   singoli, funzioni 139–146  
   singoli, verifiche 49–52  
   valore assoluto 141–142  
 VALUE 1148

- VALUE, funzione 570–571  
VAR 1148  
VAR\_POP 1148  
VAR\_SAMP 1148  
variabili  
  di binding 879  
  dichiarazione 1150  
  SQLPLUS 252–255  
  uso avanzato 287  
VARIABLE 1152  
VARIANCE, funzione 151, 1152  
vincoli  
  assegnazione di nomi 308–309  
  colonne 895  
  di controllo 308  
  dizionario dati 634  
  PRIMARY KEY 307  
  UNIQUE 305  
Virtual Private Database 349  
Virtual Storage Access Method,  
  *Vedi VSAM*  
viste  
  aggiunta ai report 244  
  creazione 63–66, 314–316  
  del dizionario dati 1153–1167  
  di gruppi 199–201  
  di sola lettura 316  
  dizionario dati 630–632  
  espansione 65–66  
  oggetti 71  
  oggetto con REF 571–576  
  oggetto, implementazione 514–520  
  oggetto, manipolazione dei dati mediante 517  
  oggetto, query 575–576  
  oggetto, uso dei trigger  
    INSTEAD OF 518–520  
  pseudocolonna User 387–389  
  selezione 689–690  
  stabilità 314–315  
  uso di alias 201  
  uso di database link 381–382  
  uso di order by 203, 315  
viste materializzate  
  aggornabili 395  
aggiornamenti automatici 405–406  
aggiornamenti manuali 406–407  
aggiornamento 402–405  
basate su chiave primaria 399–400  
basate su RowID 399–400  
create materialized view 396–402  
create materialized view log 410–413  
di sola lettura 395  
dizionario dati 647–649  
eliminazione 414  
funzionalità 393–394  
indicizzazione delle tabelle 400–401  
integrità referenziale 407–410  
log 400  
modifica 413–414  
oggetti di base 400  
percorsi di esecuzione delle query 401–402  
privilegi di sistema 394  
privilegi di tabella 394–395  
sottoprogrammi  
  di DBMS\_MVIEW 411  
sottoprogrammi  
  di DBMS OLAP 412  
VSAM 18  
VSIZE 1167
- where, clausola 178–179  
WHILE, cicli 891  
WHILE, cicli Java 588  
WHILE, cicli PL/SQL 463–465  
WIDTH\_BUCKET 1170–1171  
WRITE, funzione 556
- X**
- XML  
  attributi 773–776  
  Document Type Definition 773  
  elementi 773–776  
  inserimento, aggiornamento e annullamento con XSU 781–782  
  kit di sviluppo Oracle9iAS 730  
  personalizzazione del processo di query 783–784  
  uso di XSU 779–784  
  XML Schema 776–779  
  XMLEType 784–786, 1172  
  XSU e Java 782–783
- XSU  
  e Java 782–783  
  inserimento, aggiornamento e annullamento 781–782  
  personalizzazione del processo di query 783–784  
  uso 779–784
- W**
- Web  
  database con capacità 81–84  
  e Java 83–84  
  tecnologie e Oracle Portal 84  
  tecnologie e SQL 82–83  
when, clausola 267–271  
WHENEVER OSERROR 1168–1169  
WHENEVER SQLERROR 1169  
WHENEVER\* 1167–1168





# I MANUALI ORACLE 9i

## Gestire i vostri dati non è mai stato così facile

ORACLE 9i  
Le nuove proprietà

ROBERT G. FREEMAN



ORACLE 9i  
Sviluppo Web

BRADLEY D. BROWN



ORACLE 9i  
DBA

KEVIN LONEY  
MARLENE THERIAULT

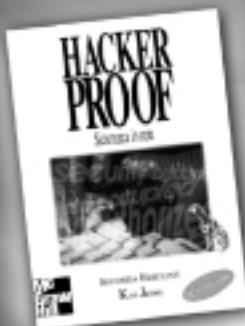
Guida a ORACLE 9i

MICHAEL ABBEY  
MIKE COREY  
IAN ADAMSON

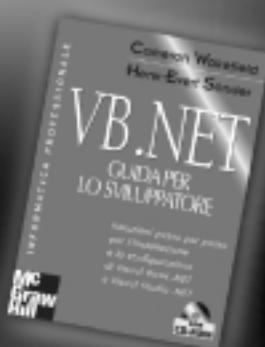


[www.mcgraw-hill.it](http://www.mcgraw-hill.it)





# CULTURA PROFESSIONALE



... RISPOSTE CONCRETE  
AI PROFESSIONISTI DELL'INFORMATICA

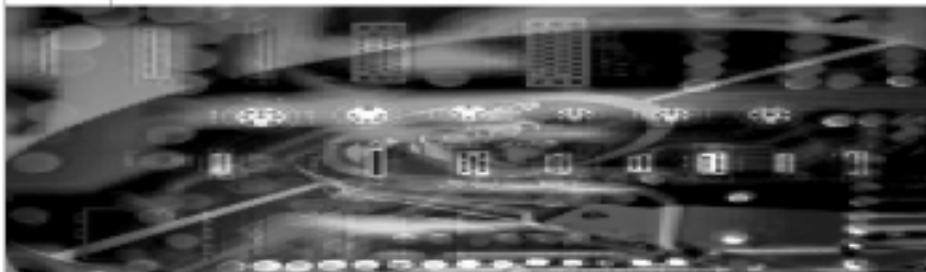
[WWW.MCGRAW-HILL.IT](http://WWW.MCGRAW-HILL.IT)

# ECDL IT Administrator - Modulo 4

 AICA Gruppo di sviluppo M4



ECDL IT Administrator - Modulo 4



# Uso avanzato delle reti

 Un riferimento  
per chi opera sulle Reti  
 Per sostenere l'esame  
ECDL IT Administrator  
Modulo 4



Programma pilota approvato dalla Fondazione ECDL

La guida di riferimento per ottenere la certificazione

Finito di stampare  
nel mese di febbraio 2003