



La guida completa



GEORGE KOCH, KEVIN LONEY

LA GUIDA COMPLETA ORACLE8

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: Paola Rossi
• REDAZIONE: Valeria Camatta
• PRODUZIONE: Gino La Rosa
• REALIZZAZIONE EDITORIALE: Infostudio – Monza
• GRAFICA DI COPERTINA: progetto di Achilli e Ghizzardi & Associati – Milano
• STAMPA: Arti Grafiche Murelli – Fizzonasco di Pieve Emanuele (MI)

Titolo originale: *Oracle8 The Complete Reference*
Edizione originale: Osborne/McGraw-Hill
Copyright © 1997 by The McGraw-Hill Companies

Copyright © 1998 McGraw-Hill Libri Italia srl
Piazza Emilia 5, 20129 Milano

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

Ogni cura è posta nella creazione, realizzazione, verifica e documentazione dei programmi contenuti in questo libro, tuttavia né l'Autore né la McGraw-Hill Libri 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. Lo stesso dicasi per ogni persona o società coinvolta nella creazione, nella produzione e nella distribuzione di questo libro.

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

Printed in Italy
1234567890AGMLLC921098
ISBN 88 386 0483-5
1^a edizione luglio 1998

Indice

Introduzione

XIII

PARTE PRIMA ● CONCETTI FONDAMENTALI DEI DATABASE

1

| | | |
|------------|---|-----------|
| Capitolo 1 | Condividere conoscenze e successo | 3 |
| 1.1 | L'approccio cooperativo | 5 |
| 1.2 | Tutti hanno dei "dati" | 6 |
| 1.3 | Il linguaggio familiare di ORACLE | 7 |
| 1.4 | Alcuni esempi comuni | 13 |
| 1.5 | Un esempio vecchio di 100 anni | 15 |
| Capitolo 2 | I pericoli di un database relazionale | 17 |
| 2.1 | È davvero facile come sembra? | 17 |
| 2.2 | Quali sono i rischi? | 18 |
| 2.3 | L'importanza della nuova visione | 19 |
| 2.4 | Codici, abbreviazioni e standard di denominazione | 20 |
| 2.5 | Perché vengono utilizzati i codici e non la lingua corrente? | 22 |
| 2.6 | Come arginare la confusione | 23 |
| 2.7 | Maiuscole e minuscole nei nomi e nei dati | 35 |
| 2.8 | Normalizzazione dei nomi | 35 |
| 2.9 | Cogliere l'opportunità | 36 |

PARTE SECONDA • SQL: DA PRINCIPIANTI A ESPERTI 37

| | | |
|------------|--|------------|
| Capitolo 3 | Le parti fondamentali del discorso in SQL | 39 |
| 3.1 | Stile | 41 |
| 3.2 | Utilizzo di SQL per selezionare dati da tabelle | 42 |
| 3.3 | I comandi select, from, where e order by | 45 |
| 3.4 | Logica e valore | 47 |
| 3.5 | LIKE | 51 |
| 3.6 | Un altro impiego delle sottoquery con where | 58 |
| 3.7 | Come combinare le tabelle | 62 |
| 3.8 | Creazione di una vista | 64 |
| Capitolo 4 | Elementi di base dei database relazionali a oggetti | 69 |
| 4.1 | È obbligatorio utilizzare gli oggetti? | 69 |
| 4.2 | Perché utilizzare gli oggetti? | 70 |
| 4.3 | Tutti possiedono degli oggetti | 71 |
| 4.4 | Un esempio di oggetto comune | 77 |
| 4.5 | Analisi e progettazione orientata agli oggetti | 84 |
| 4.6 | I prossimi capitoli | 85 |
| Capitolo 5 | Report e comandi fondamentali di SQLPLUS | 87 |
| 5.1 | Creazione di un report semplice | 90 |
| 5.2 | Altre caratteristiche | 101 |
| 5.3 | Controllo dell'ambiente SQLPLUS | 108 |
| 5.4 | I fondamenti | 110 |
| Capitolo 6 | Ottenere e modificare informazioni di testo | 113 |
| 6.1 | I tipi di dati | 114 |
| 6.2 | Che cos'è una stringa? | 114 |
| 6.3 | Notazione | 115 |
| 6.4 | Concatenazione () | 117 |
| 6.5 | Come tagliare e incollare le stringhe | 118 |
| 6.6 | Le funzioni stringa order by e where with | 134 |
| 6.7 | Riepilogo | 138 |
| Capitolo 7 | Giocare con i numeri | 139 |
| 7.1 | Le tre classi delle funzioni numeriche | 139 |
| 7.2 | Notazione | 141 |
| 7.3 | Funzioni a valori singoli | 142 |
| 7.4 | Funzioni di gruppo | 150 |
| 7.5 | Funzioni di elenco | 157 |
| 7.6 | Ricerca di righe con MAX o MIN | 158 |
| 7.7 | Precedenza e parentesi | 159 |
| 7.8 | Riepilogo | 161 |

| | | |
|-------------|---|------------|
| Capitolo 8 | Le date | 163 |
| 8.1 | Aritmetica delle date | 163 |
| 8.2 | ROUND e TRUNC in calcoli di date | 172 |
| 8.3 | Formattazione di TO_DATE e TO_CHAR | 173 |
| 8.4 | Date nelle clausole where | 182 |
| 8.5 | L'anno 2000 | 184 |
| Capitolo 9 | Funzioni di conversione e trasformazione | 187 |
| 9.1 | Funzioni elementari di conversione | 189 |
| 9.2 | Funzioni di conversione specializzate | 195 |
| 9.3 | Funzioni di trasformazione | 195 |
| 9.4 | Riepilogo | 198 |
| Capitolo 10 | Raggruppamento di righe | 199 |
| 10.1 | Utilizzo di group by e having | 201 |
| 10.2 | Viste di gruppi | 205 |
| 10.3 | La potenza delle viste di gruppi | 209 |
| 10.4 | where, having, group by e order by | 213 |
| Capitolo 11 | Una query dipendente da un'altra | 215 |
| 11.1 | Sottoquery avanzate | 215 |
| 11.2 | UNION, INTERSECT e MINUS | 228 |
| Capitolo 12 | Alcune possibilità complesse | 241 |
| 12.1 | Creazione di una vista complessa | 241 |
| 12.2 | Alberi genealogici e connect by | 246 |
| 12.3 | Utilizzo di viste nella clausola from | 255 |
| Capitolo 13 | Creazione di un report in SQLPLUS | 257 |
| 13.1 | Formattazione avanzata | 257 |
| 13.2 | set termout off e set termout on | 272 |
| 13.3 | Variabili in SQLPLUS | 272 |
| 13.4 | Formattazione di numeri | 275 |
| 13.5 | Utilizzo di mask.sql | 278 |
| 13.6 | Utilizzo di buffer per salvare comandi SQLPLUS | 278 |
| 13.7 | show all e spool | 281 |
| 13.8 | Aggiunta di righe vuote | 281 |
| 13.9 | Ulteriori controlli per la realizzazione di report | 283 |
| Capitolo 14 | Modifica dei dati: insert, update e delete | 285 |
| 14.1 | insert | 285 |
| 14.2 | rollback, commit e autocommit | 288 |
| 14.3 | delete | 291 |
| 14.4 | update | 292 |

| | | |
|-------------|--|------------|
| Capitolo 15 | Utilizzo avanzato di funzioni e variabili | 297 |
| 15.1 | Funzioni in order by | 297 |
| 15.2 | Diagrammi a barre e grafici | 298 |
| 15.3 | Utilizzo di TRANSLATE | 300 |
| 15.4 | Copia e incollamento complessi | 304 |
| 15.5 | Conteggio delle ricorrenze di stringhe in stringhe più lunghe | 308 |
| 15.6 | Variabili e sostituzioni registrate | 309 |
| Capitolo 16 | La funzione DECODE | 315 |
| 16.1 | if, then, else | 315 |
| 16.2 | Esempio: datazione di fatture | 316 |
| 16.3 | Trasposizione di una tabella | 321 |
| 16.4 | Utilizzo di MOD in DECODE | 323 |
| 16.5 | order by e RowNum | 325 |
| 16.6 | Colonne e calcoli in then ed else | 327 |
| 16.7 | Maggiore, minore e uguale in DECODE | 329 |
| Capitolo 17 | Creazione, scaricamento e modifica di tabelle e viste | 331 |
| 17.1 | Creazione di una tabella | 331 |
| 17.2 | Scaricamento di tabelle | 339 |
| 17.3 | Modifica di tabelle | 339 |
| 17.4 | Creazione di una vista | 341 |
| 17.5 | Creazione di una tabella a partire da un'altra | 345 |
| 17.6 | Creazione di una tabella di solo indice | 346 |
| 17.7 | Utilizzo di tabelle partizionate | 347 |
| Capitolo 18 | ORACLE e l'autorità | 353 |
| 18.1 | Utenti, ruoli e privilegi | 353 |
| 18.2 | Che cosa possono concedere gli utenti | 357 |
| 18.3 | Concessione di risorse limitate | 370 |
| Capitolo 19 | Modifica dell'ambiente di ORACLE | 371 |
| 19.1 | Indici | 371 |
| 19.2 | Tablespace e struttura del database | 380 |
| 19.3 | Cluster | 385 |
| 19.4 | Sequenze | 386 |
| Capitolo 20 | SQLPLUS | 389 |
| 20.1 | Generazione di codice per una query | 389 |
| 20.2 | Caricamento di variabili | 395 |
| 20.3 | Creazione e annidamento di file di avvio e comandi | 397 |
| 20.4 | Riepilogo | 401 |

| | | |
|-------------|--|------------|
| Capitolo 21 | Accesso a dati remoti | 403 |
| 21.1 | Link di database | 403 |
| 21.2 | Utilizzo di sinonimi per la trasparenza di locazione | 410 |
| 21.3 | Utilizzo della pseudocolonna User nelle viste | 411 |
| 21.4 | Collegamenti dinamici: utilizzo del comando copy di SQLPLUS | 413 |
| 21.5 | Connessione a un database remoto | 415 |
| 21.6 | Strumenti di gestione: Oracle*Names | 416 |
| Capitolo 22 | Introduzione a PL/SQL | 419 |
| 22.1 | PL/SQL: nozioni di base | 419 |
| 22.2 | La sezione delle dichiarazioni | 420 |
| 22.3 | La sezione dei comandi eseguibili | 423 |
| 22.4 | La sezione per la gestione delle eccezioni | 434 |
| Capitolo 23 | I trigger | 437 |
| 23.1 | Privilegi di sistema necessari | 437 |
| 23.2 | Privilegi di tabella richiesti | 438 |
| 23.3 | Tipi di trigger | 438 |
| 23.4 | Sintassi dei trigger | 440 |
| 23.5 | Attivazione e disattivazione dei trigger | 447 |
| 23.6 | Sostituzione di trigger | 449 |
| 23.7 | Scaricamento di trigger | 449 |
| Capitolo 24 | Le procedure | 451 |
| 24.1 | Privilegi di sistema necessari | 452 |
| 24.2 | Privilegi di tabella necessari | 454 |
| 24.3 | Procedure e funzioni | 454 |
| 24.4 | Procedure e package | 454 |
| 24.5 | Sintassi del comando create procedure | 455 |
| 24.6 | Sintassi del comando create function | 456 |
| 24.7 | Sintassi per il comando create package | 463 |
| 24.8 | Visualizzazione del codice sorgente di oggetti procedurali esistenti | 467 |
| 24.9 | Compilazione di procedure, funzioni e package | 467 |
| 24.10 | Sostituzione di procedure, funzioni e package | 468 |
| 24.11 | Scaricamento di procedure, funzioni e package | 469 |
| Capitolo 25 | Implementazione di tipi, viste oggetto e metodi | 471 |
| 25.1 | Nuova analisi dei tipi di dati astratti | 471 |
| 25.2 | Implementazione di viste oggetto | 476 |
| 25.3 | Metodi | 483 |

| | | |
|-------------|---|------------|
| Capitolo 26 | Tabelle annidate e array variabili | 489 |
| 26.1 | Array variabili | 489 |
| 26.2 | Tabelle annidate | 496 |
| 26.3 | Problemi nella gestione di tabelle annidate e array variabili | 503 |
| Capitolo 27 | Utilizzo di LOB in ORACLE8 | 507 |
| 27.1 | Tipi di dati disponibili | 507 |
| 27.2 | Definizione dei parametri di memorizzazione per i dati LOB | 509 |
| 27.3 | Gestione e selezione dei valori LOB | 511 |
| Capitolo 28 | Gli snapshot | 531 |
| 28.1 | Che cosa sono gli snapshot? | 532 |
| 28.2 | Privilegi di sistema richiesti | 532 |
| 28.3 | Privilegi richiesti per la tabella | 533 |
| 28.4 | Snapshot semplici e snapshot complessi | 533 |
| 28.5 | Snapshot di sola lettura e snapshot aggiornabili | 534 |
| 28.6 | Sintassi del comando create snapshot | 535 |
| 28.7 | Aggiornamento degli snapshot | 541 |
| 28.8 | Snapshot e trigger | 547 |
| 28.9 | Sintassi per il comando create snapshot log | 548 |
| 28.10 | Visualizzazione di informazioni su snapshot esistenti | 549 |
| 28.11 | Modifica di snapshot e log di snapshot | 551 |
| 28.12 | Scaricamento di snapshot e log di snapshot | 552 |
| Capitolo 29 | Utilizzo di ConText per ricerche di testo | 555 |
| 29.1 | Aggiunta di testo al database | 555 |
| 29.2 | Query di testo dal database | 557 |
| 29.3 | Impostazione dell'opzione ConText | 570 |
| Capitolo 30 | Impostazione dell'opzione ConText | 571 |
| 30.1 | Impostazione del database per le ricerche di testo | 571 |
| 30.2 | Impostazione della tabella per query ConText | 575 |
| 30.3 | Ottimizzazione degli indici testuali | 583 |
| 30.4 | Query in due passaggi | 584 |
| 30.5 | Utilizzo dei servizi linguistici | 586 |
| Capitolo 31 | Concetti orientati agli oggetti avanzati in ORACLE8 | 589 |
| 31.1 | Oggetti riga e oggetti colonna | 589 |
| 31.2 | Tabelle oggetto e OID | 590 |
| 31.3 | Viste oggetto con REF | 598 |
| 31.4 | PL/SQL a oggetti | 603 |
| 31.5 | Oggetti nel database | 605 |

PARTE TERZA • IL DIZIONARIO DI DATI DI ORACLE8 607

| | | |
|-------------|---|------------|
| Capitolo 32 | Guida al dizionario di dati di ORACLE8 | 609 |
| 32.1 | Nota sulla nomenclatura | 610 |
| 32.2 | Le “carte stradali”: DICTIONARY (DICT) e DICT_COLUMNS | 610 |
| 32.3 | Oggetti da cui è possibile selezionare: tabelle (e colonne), viste, sinonimi e sequenze | 612 |
| 32.4 | Vincoli e commenti | 622 |
| 32.5 | Indici e cluster | 627 |
| 32.6 | Oggetti specifici di ORACLE8 | 631 |
| 32.7 | Link di database e snapshot | 635 |
| 32.8 | Trigger, procedure, funzioni e package | 639 |
| 32.9 | Allocazione e utilizzo dello spazio, compreso il partizionamento | 642 |
| 32.10 | Utenti e privilegi | 648 |
| 32.11 | Ruoli | 651 |
| 32.12 | Revisioni | 652 |
| 32.13 | Monitoraggio: le tabelle V\$ o tabelle di prestazioni dinamiche | 654 |
| 32.14 | Varie | 655 |

PARTE QUARTA • OTTIMIZZAZIONE DEL PROGETTO 659

| | | |
|-------------|---|------------|
| Capitolo 33 | L'importanza del fattore umano | 661 |
| 33.1 | Comprensione dei compiti dell'applicazione | 662 |
| 33.2 | Comprensione dei dati | 667 |
| 33.3 | Il modello commerciale | 670 |
| 33.4 | Inserimento dei dati | 670 |
| 33.5 | Query e report | 671 |
| 33.6 | Conclusioni | 671 |
| Capitolo 34 | Prestazioni e progettazione | 673 |
| 34.1 | Denormalizzazione e integrità dei dati | 674 |
| 34.2 | La tabella dei calcoli | 682 |
| 34.3 | Riepilogo | 682 |
| Capitolo 35 | I dieci comandamenti della progettazione | 683 |
| 35.1 | Verso la normalizzazione dei nomi degli oggetti | 683 |
| 35.2 | Sinonimi di nomi di oggetti | 690 |
| 35.3 | Chiavi intelligenti e valori di colonna | 690 |
| 35.4 | Comandamenti | 691 |

| | | |
|--|--|-------------|
| Capitolo 36 | Guida all'ottimizzatore di ORACLE | 693 |
| 36.1 | Quale ottimizzatore? | 693 |
| 36.2 | Operazioni di accesso alle tabelle | 694 |
| 36.3 | Operazioni che utilizzano indici | 696 |
| 36.4 | Operazioni che gestiscono insiemi di dati | 706 |
| 36.5 | Operazioni di unione | 720 |
| 36.6 | Visualizzazione del percorso di esecuzione | 732 |
| 36.7 | Operazioni varie | 738 |
| 36.8 | Riepilogo | 744 |
| PARTE QUINTA ● RIFERIMENTO ALFABETICO | | 745 |
| Capitolo 37 | Guida di riferimento in ordine alfabetico | 747 |
| 37.1 | Contenuti della guida di riferimento | 747 |
| 37.2 | Che cosa non contiene la guida di riferimento | 747 |
| 37.3 | Formato generale delle voci | 748 |
| 37.4 | Ordine dell'elenco | 750 |
| PARTE SESTA ● APPENDICI | | 1031 |
| Appendice A | Le tabelle utilizzate nel libro | 1033 |
| A.1 | Utilizzo delle tabelle | 1033 |
| Indice analitico | | 1071 |

Introduzione

ORACLE è il database più diffuso del mondo. Funziona su praticamente tutti i tipi di computer, da PC e Macintosh fino ai minicomputer e ai mainframe. In tutte queste macchine il funzionamento è virtualmente identico, perciò basta 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, infatti comprende oltre cinquanta volumi. *La guida completa ORACLE8* 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 guida di riferimento che tutti gli utenti e gli sviluppatori dovrebbero tenere a portata di mano.

Le persone a 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, 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'attività. Il linguaggio di questo libro è semplice e chiaro, privo di termini gergali. Le conoscenze di computer o database richieste come presupposto al lettore sono assai poche. 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'utilizzo di SQLPLUS come generatore di codice, l'ereditarietà e CONNECT BY, l'eliminazione di NOT IN con un join esterno, l'utilizzo di ConText, l'implementazione delle opzioni relazionali a oggetti e molti altri. Nel testo sono inoltre chiariti molti aspetti che spesso sono oggetto di confusione e suggeriti rigorosi principi guida per convenzioni di denominazione, tecniche di sviluppo delle applicazioni, progettazione e prestazioni.

Per qualsiasi utente e sviluppatore, oltre duecento pagine del libro sono dedicate a una completa guida di riferimento alfabetica che contiene tutti i principali aspetti, comandi, funzioni e caratteristiche di ORACLE, con sintassi, riferimenti incrociati ed esempi. Si tratta della più ampia guida di riferimento pubblicata su questo argomento.

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 di 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 la generazione di codice, il caricamento di variabili e l'annidamento di file di avvio e processi host in SQLPLUS, oltre ai modi per sfruttare le caratteristiche dei database distribuiti, trigger e procedure memorizzate, e le altre funzionalità di ORACLE8.

Sono inoltre trattate 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 terza fornisce una guida orientata all'utente per il dizionario di dati di ORACLE8, l'equivalente delle Pagine Gialle nel mondo dei database. Invece di elencare in ordine alfabetico le viste del dizionario di dati, queste vengono raggruppate per argomento, in modo da abbreviare i tempi richiesti per scoprire quale vista fa al caso proprio. Diversi esempi concreti illustrano l'uso appropriato di queste viste.

Nella Parte quarta sono trattati argomenti vitali nella progettazione di applicazioni utili e ben accolte. Gli strumenti di ORACLE offrono una notevole opportunità di creare applicazioni efficaci e apprezzate dagli utenti, ma molti sviluppatori non conoscono alcune delle metodiche utilizzabili.

Questa parte si rivolge specificamente agli sviluppatori, le persone che hanno la responsabilità di comprendere un'attività e progettare un programma ORACLE in grado di soddisfarla. Gli argomenti non dovrebbero risultare del tutto incomprensibili per gli utenti finali, ma si presuppone che il lettore disponga di nozioni tecniche riguardo l'elaborazione di dati ed esperienza nello sviluppo di applicazioni per computer.

Lo scopo è quello di discutere le tecniche di sviluppo con ORACLE che si sono dimostrate efficaci e utili per gli utenti finali, oltre che di proporre alcuni nuovi approcci alla progettazione in aree finora ignorate. Questa parte comprende "I dieci comandamenti della progettazione", un elenco di tutte le regole vitali per il processo di sviluppo, e si conclude con una guida orientata all'utente per l'ottimizzatore di ORACLE.

La Parte quinta contiene il riferimento completo per il server ORACLE. Le pagine introduttive sono molto utili per comprendere meglio il testo. Questa parte contiene riferimenti per la maggior parte dei comandi, delle parole chiavi, dei prodotti, delle caratteristiche e delle funzionalità di ORACLE, con molti rimandi ai vari argomenti. La guida si rivolge sia agli sviluppatori sia agli utenti di ORACLE, ma presuppone una certa familiarità con il prodotto. 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.

La Parte sesta contiene 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.

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.

Breve panoramica sui capitoli

Il Capitolo 1 spiega i concetti di base del modello relazionale e le opportunità di sviluppare applicazioni di successo condividendo le responsabilità della progettazione con utenti finali dotati di buone capacità.

Nel Capitolo 2 sono descritti alcuni dei rischi inerenti lo sviluppo in un ambiente relazionale con tempi rapidi e sono discussi molti degli errori classici commessi nel passato. In questo capitolo sono inoltre presentati i metodi di controllo dei rischi.

Il Capitolo 3 è un'introduzione al linguaggio SQL, con lo stile utilizzato in questo libro, logica e valore, sottoquery, unione di tabelle e creazione di viste.

Nel Capitolo 4 è presentato il concetto di tipo di dati astratto, il fondamento dei database relazionali a oggetti.

Il Capitolo 5 mostra come creare un semplice report in SQLPLUS, come utilizzare un editor, servirsi dei comandi di SQLPLUS e capire le differenze tra SQLPLUS e SQL.

Nel Capitolo 6 viene introdotto il concetto di stringa di caratteri, sono esaminate le funzioni sui caratteri e descritte le differenze tra caratteri, numeri e date in ORACLE.

Nel Capitolo 7 si esplorano i numeri e le relative funzioni, tra cui quelle a valore singolo, di gruppo e di elenco.

Il Capitolo 8 mostra come utilizzare le eccezionali funzioni per la manipolazione delle date di ORACLE, come calcolare le differenze tra le date e come formattare le date per la visualizzazione.

Nel Capitolo 9 sono trattate le funzioni di conversione da un tipo di dati in un altro, o che "trasformano" i dati in una forma diversa da quella in cui appaiono normalmente.

Il Capitolo 10 mostra come raggruppare e riepilogare le informazioni in ORACLE e come creare viste di dati riepilogati.

Nel Capitolo 11 sono trattate le sottoquery avanzate, le “sottoquery correlate”, la tecnica del join esterno e l’uso degli operatori UNION, INTERSECT e MINUS per combinare tabelle di ORACLE. Questo capitolo contiene inoltre una trattazione avanzata della precedenza e la spiegazione di come utilizzare i join esterni e NOT EXISTS per sostituire l’operatore NOT IN.

Il Capitolo 12 mostra come creare viste molto complesse e contiene un esteso esame dell’ereditarietà e di CONNECT BY.

Nel Capitolo 13 sono trattate tecniche avanzate per la creazione di report e la formattazione in SQLPLUS, per calcolare medie pesate, per utilizzare variabili nei titoli, per formattare i numeri e per utilizzare varie istruzioni SQL nei report.

Il Capitolo 14 discute il modo in cui si modificano i dati nelle tabelle di ORACLE e l’importanza dei processi di commit e rollback.

Il Capitolo 15 introduce i metodi per creare grafici e diagrammi a barre, per inserire virgole nei numeri, per eseguire complesse funzioni di taglia e incolla e per caricare dinamicamente le variabili.

Nel Capitolo 16 si approfondisce l’uso della potente funzione DECODE in ORACLE, spiegando come effettuare la trasposizione di una tabella, come dare le fatture, come controllare il movimento della carta nelle stampanti e come regolare i commit per ampi gruppi di record.

Il Capitolo 17 tratta in dettaglio le basi della manipolazione di tabelle, spiegando come specificare vincoli e partizioni e descrivendo le viste di sistema che contengono informazioni sulle tabelle.

Nel Capitolo 18 sono trattate le funzioni di sicurezza di ORACLE, tra cui l’autorità dei DBA e i privilegi di accesso che possono essere concessi a ogni utente, o che gli utenti possono concedere ad altri. Viene inoltre spiegato come le modifiche nell’autorità influenzino gli utenti che dipendono da altri e l’uso dei ruoli nelle applicazioni e nell’amministrazione dei database.

Nel Capitolo 19 sono trattati gli indici, l’uso dello spazio, la struttura del database, le tecniche per la gestione delle tabelle mediante i cluster, le sequenze e i termini tecnici di base per ORACLE.

Il Capitolo 20 tratta materiale avanzato per sviluppatori e utenti esperti. Mostra metodi per l’utilizzo di SQLPLUS al fine di generare codice, caricare variabili, creare viste con numero di tabelle variabile e utilizzare processi host in modo interattivo.

Il Capitolo 21 descrive i metodi utilizzati per accedere a dati che si trovano in database remoti e mostra come creare e mantenere link di database.

Nel Capitolo 22 vengono presentate le strutture e le caratteristiche delle estensioni al linguaggio SQL di ORACLE. Comprendere PL/SQL è fondamentale per sfruttare caratteristiche come package, procedure e metodi.

Il Capitolo 23 descrive i tipi di trigger disponibili in ORACLE, con esempi dei trigger utilizzati più comunemente.

Nel Capitolo 24 viene mostrato come utilizzare procedure, package e funzioni in ORACLE e come creare, compilare, eseguire il debugging e manipolare tali oggetti.

Il Capitolo 25 mostra i passaggi necessari per implementare i concetti relazionali a oggetti descritti per la prima volta nel Capitolo 4. Gli esempi illustrano come sovrapporre strutture relazionali a oggetti alle tabelle relazionali esistenti.

Il Capitolo 26 mostra come implementare tabelle annidate e array variabili nelle applicazioni e descrive i problemi di gestione relativi.

Nel Capitolo 27 viene mostrato come utilizzare i tipi di dati LOB disponibili a partire da ORACLE8 e come manipolare i valori memorizzati nelle colonne di tipo LOB.

Il Capitolo 28 mostra come utilizzare gli oggetti snapshot di ORACLE per gestire la replicazione di dati remoti. Spiega inoltre come funzionano gli snapshot, gli oggetti creati nei database locali e remoti, e come gestire la programmazione della replicazione.

Il Capitolo 29 mostra come eseguire ricerche di testo (come le espansioni della radice, le corrispondenze fuzzy e la ricerca di frasi).

Nel Capitolo 30 viene spiegato come configurare le tabelle in modo da poter utilizzare i metodi di ricerca descritti nel Capitolo 29. Questo capitolo si rivolge agli amministratori delle applicazioni che necessitano di configurare e gestire indici di testo.

Il Capitolo 31 descrive oggetti riga e riferimenti, oltre alle estensioni a oggetti per PL/SQL. I concetti presentati in questo capitolo sono utili per gli sviluppatori avanzati che desiderano implementare caratteristiche orientate agli oggetti in un database di ORACLE.

Nel Capitolo 32 sono mostrate le viste del dizionario di dati utilizzate con maggiore frequenza, in maniera orientata all'utente: le viste sono organizzate per funzionalità in modo da facilitarne la localizzazione.

Il Capitolo 33 offre una guida sicura alla creazione di applicazioni che abbiano un chiaro significato per l'utente e svolgano bene il compito a cui sono dedicate. Questo approccio è fondamentale per avere un buon successo nello sviluppo di applicazioni.

Nel Capitolo 34 sono esaminati alcuni dei problemi e delle lacune circa la normalizzazione e i metodi di progettazione; sono inoltre forniti suggerimenti per migliorare le prestazioni e la progettazione di applicazioni ORACLE.

Il Capitolo 35 presenta alcuni concetti nuovi per la comunità relazionale, con lo scopo di portare un passo oltre la metodologia di progettazione; si tratta di integrità dei nomi, normalizzazione dei nomi di oggetti e singolarità. Vengono inoltre trattate aree che per tradizione risultano irte di difficoltà e si conclude con le dieci regole di base per la progettazione.

Il Capitolo 36 spiega il modo in cui ORACLE accede ai dati e sceglie il percorso da utilizzare per l'accesso. Vengono inoltre mostrate le conseguenze di queste scelte e i comandi che consentono di prevedere e influenzare la scelta del percorso.

Il Capitolo 37 è una guida di riferimento completa, in ordine alfabetico, di comandi, parole chiave, funzioni e altri elementi di ORACLE.

Nell'Appendice A sono inclusi i listati completi per tutte le tabelle utilizzate nel libro.

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 alfabetico del Capitolo 37 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 grassetto non viene utilizzato nei listati di esempio.
- select, from, where, order by, having e group by sono riportati in 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; questo vale ad esempio per 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, LOCAZIONE e così via, sono riportati in maiuscolo.

Nel Capitolo 3 fornita un’introduzione allo stile utilizzato per i capitoli del libro fino al Capitolo 36. Il Capitolo 37 contiene un’importante testo introduttivo che descrive le convenzioni di stile adottate e andrebbe letto con cura prima di utilizzare gli elenchi alfabetici.

- 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**

• 1.5 **Un esempio vecchio di 100 anni**

Per poter creare e utilizzare rapidamente ed efficacemente un'applicazione ORACLE8, 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, possono essere sviluppate 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 diversi, 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, tese e abnormi. Si instaurano le barriere e le procedure artificiali che prendono le mosse da questo isolazionismo, contribuendo ad aggravare la "sindrome".

Va bene, penserà il lettore, 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, da 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 sul modello relazionale di facile comprensione (spiegato tra breve); in questo modo i non esperti comprendevano prontamente che cosa faceva ORACLE e come lo faceva. Ciò lo rese di facile approccio.

Inoltre ORACLE venne progettato per essere eseguito virtualmente 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 producevano hardware "proprietario", sistemi operativi "proprietari", database "proprietari" e applicazioni "proprietarie", ORACLE consentiva agli utenti che lo utilizzavano 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 poteva essere eseguito su qualsiasi tipo di computer posseduto dall'azienda. 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 hanno ancora accettato o compreso, né hanno capito l'importanza vitale del fatto che le vecchie e artificiali barriere tra "utenti" e "sistemi" continuano 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. Riguardo le modalità di progettazione e gestione dateate 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 le persone 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. Potrebbe sembrare che questa trattazione sia esclusivamente a beneficio degli "utenti" e 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 di queste 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, comprendere i 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. Con ORACLE, si ottiene 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. Poiché 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 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 vengono selezionati e manipolati i dati per rivelare 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 modificandoli. Ciò viene definito come *elaborazione* e, fondamentalmente, coinvolge le tre fasi con cui vengono organizzate le informazioni.

Tutte queste operazioni potrebbero essere effettuate con una scatola di sigari, carta e matita, ma con l'aumentare del volume dei dati, tendenzialmente cambiano anche gli strumenti. Si possono utilizzare un archivio, calcolatori, 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) come ORACLE consente di eseguire questi compiti in maniera comprensibile e ragionevolmente semplice. In pratica con ORACLE è possibile eseguire tre compiti:

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

Nella Figura 1.1 viene illustrata questa semplicità d'azione.

ORACLE supporta questo approccio in tre fasi e fornisce strumenti intelligenti che consentono un notevole livello qualitativo nelle modalità con cui i dati vengono

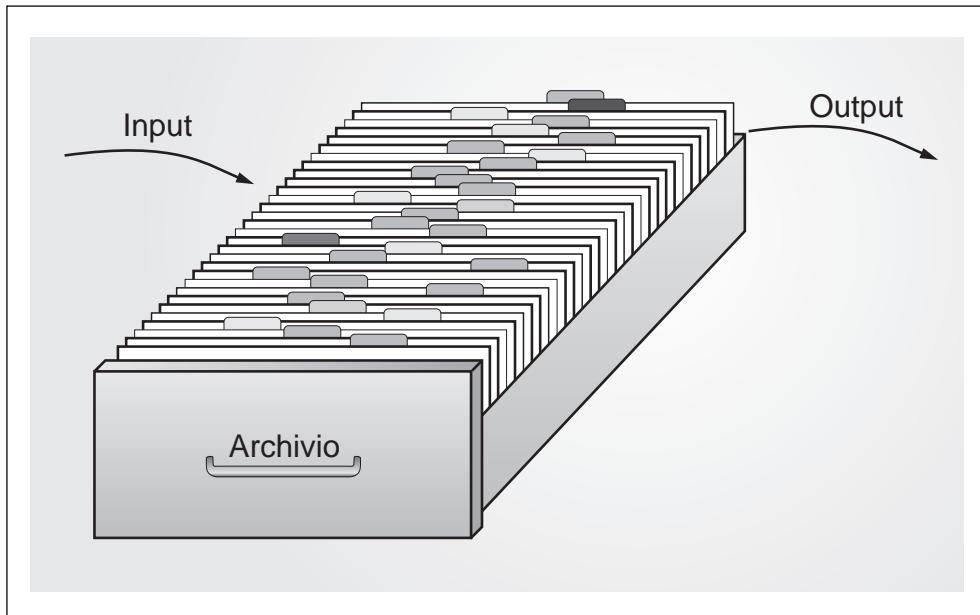


Figura 1.1 Gestione dei dati in ORACLE.

reperiti, visualizzati, modificati e inseriti, mantenuti al sicuro ed estrapolati per manipolarli e costruire report sulla base delle informazioni che contengono.

In un sistema di gestione di database relazionale a oggetti (ORDBMS) vengono estese le possibilità di un RDBMS per supportare concetti orientati agli oggetti. ORACLE può essere utilizzato come un RDBMS per trarre vantaggio delle 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 orizzontale 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 tabella in un database di ORACLE sono le stesse utilizzate nelle conversazioni quotidiane. Non si tratta di termini dal significato particolare, insolito o esoterico: significano esattamente ciò che sembrano.

| 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.

Tabelle di informazioni

Le informazioni in ORACLE sono memorizzate sotto forma di tabelle, come è mostrato nella Figura 1.3. In ciascuna di queste tabelle sono presenti una o più colonne. Le intestazioni come Città, Temperatura, Umidità e Condizione 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, un'intestazione di 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 compresi 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 incomprensione che viene prodotto dall’utilizzo di gergo tecnico non necessario. Come ORACLE, 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. Ciò consentì agli utenti finali di estrarre le informazioni in modo autonomo, senza rivolgersi ai sistemisti per ogni piccolo report.

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

Nome tabella
CLIMA

Colonna
Temperatura

| 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.

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 delle condizioni climatiche 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 con cui lo fa un operatore e con 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 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
```

E 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
```

In ORACLE appare immediatamente la risposta:

| 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; in seguito viene spiegato come specificare l'ordinamento).

Con gli strumenti di query di ORACLE si possono impostare molte altre domande, tuttavia gli esempi riportati dimostrano quanto sia facile ricavare informazioni da un database di ORACLE nella forma più corrispondente alle esigenze dell'utente. Si possono costruire richieste complesse partendo da elementi semplici, ma il metodo utilizzato rimane sempre facilmente comprensibile. Ad 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
```

(> significa "maggiore di")

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
```

(< significa "minore di")

la risposta sarebbe:

| Citta | Temperatura | Umidita |
|--------|-------------|---------|
| PARIGI | 27 | 62 |

Che cosa significa relazionale

Si osservi 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 LOCAZIONE separata per le città e i rispettivi paesi, come è illustrato nella Figura 1.4.

Per ciascuna città nella tabella CLIMA è possibile semplicemente dare un'occhiata alla tabella LOCAZIONE, trovare il nome nella colonna Citta, 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 hanno un elemento significativo in comune: la colonna Citta. A ogni nome di città nella tabella CLIMA corrisponde un nome identico nella tabella LOCAZIONE.

| CLIMA | | | |
|------------|-------------|---------|------------|
| Citta | 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 |

| LOCAZIONE | |
|------------|-------------|
| Citta | 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 LOCAZIONE.

Ad 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.

Come si risolve questo quesito? È presente una sola voce AUSTRALIA, nella colonna Paese, nella tabella LOCAZIONE. Vicino a questa, nella colonna Città della stessa riga è visualizzato il nome della città, SYDNEY. Si cerca questo nome nella colonna Città 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: in nome di città in una tabella è correlato a un nome di città nell'altra. Il termine *database relazionale* è riferito a questo tipo di relazione. Si veda la Figura 1.5. 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 di loro se hanno una colonna con un tipo di informazione comune.

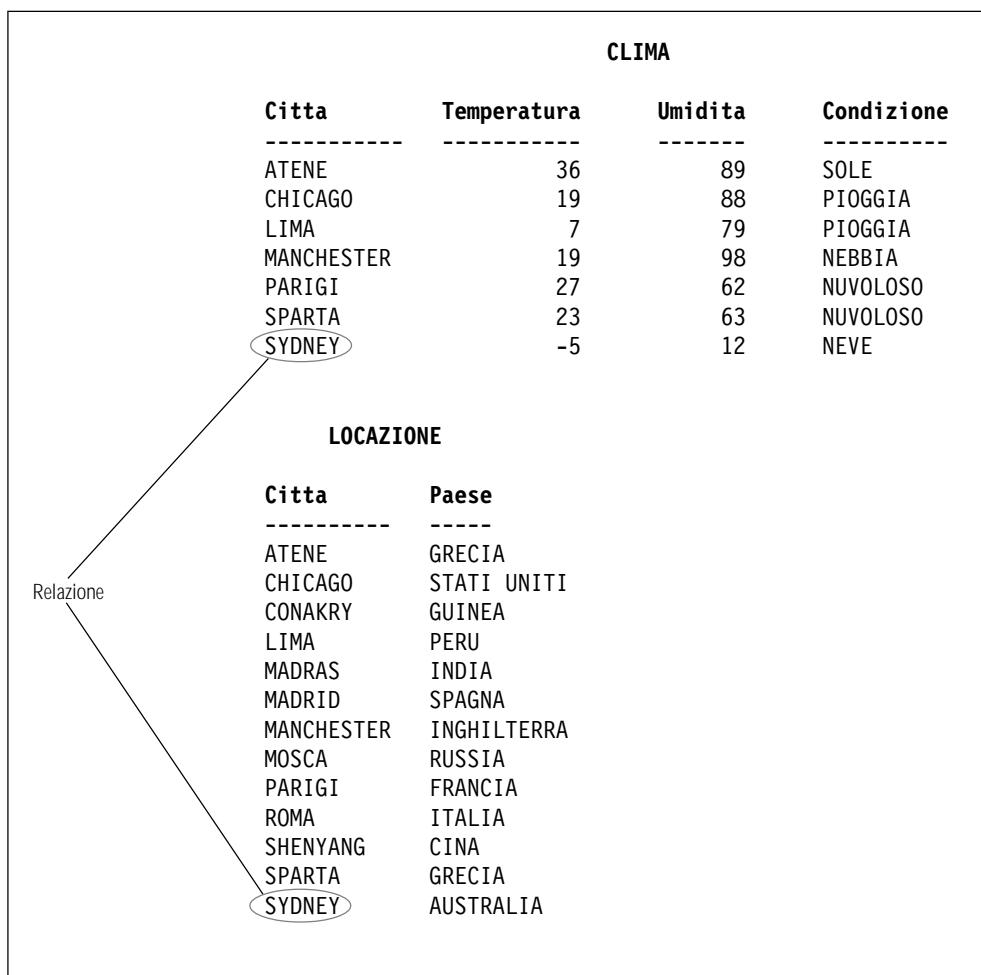


Figura 1.5 La relazione tra le tabelle CLIMA e LOCAZIONE.

1.4 Alcuni esempi comuni

Una volta compreso il concetto fondamentale di database relazionale, si vedono tavole, 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 più alto numero 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. Nella Figura 1.7 sono riportati i risultati di alcune squadre di hockey. Qual è la classifica di tutte le squadre? Quali squadre hanno giocato più partite? Ottenere le risposte a queste domande con ORACLE è facile: basta impostare delle semplici query.

| 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 del mercato azionario.

| RISULTATI COMPLESSIVI | | | |
|------------------------------|--------------|--------------|-------------|
| Squadra | Vinte | Perse | Pari |
| Boston..... | 17 | 13 | 3 |
| Buffalo..... | 21 | 9 | 4 |
| Calgary..... | 14 | 11 | 9 |
| Chicago..... | 19 | 13 | 2 |
| Detroit..... | 10 | 18 | 5 |
| Edmonton..... | 16 | 11 | 7 |
| Hartford..... | 16 | 17 | 1 |
| Los Angeles..... | 16 | 14 | 3 |
| Minnesota..... | 17 | 15 | 2 |
| Montreal..... | 20 | 13 | 4 |
| New Jersey..... | 15 | 15 | 3 |
| N.Y. Rangers..... | 15 | 14 | 5 |
| N.Y. Islanders..... | 11 | 20 | 4 |
| Philadelphia..... | 16 | 14 | 4 |
| Pittsburgh..... | 13 | 16 | 3 |
| Quebec..... | 6 | 23 | 4 |
| St Louis..... | 14 | 12 | 6 |
| Toronto..... | 16 | 18 | 0 |
| Vancouver..... | 11 | 16 | 6 |
| Washington..... | 13 | 15 | 4 |
| Winnipeg..... | 14 | 13 | 5 |

Figura 1.7 Una tabella di risultati di hockey.

Nella Figura 1.8 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.

| 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 |
| Film | B | 4 |
| Notizie | A | 1 |
| Necrologi | F | 6 |
| Sport | D | 1 |
| Televisione | B | 7 |
| Meteo | C | 2 |

Figura 1.8 Una tabella basata sulle sezioni di un giornale.

1.5 Un esempio vecchio di 100 anni

Un antico e deteriorato libro mastro, che porta la data iniziale del 1896, appartenuto a un tale G. B. Talbot, contiene le voci riportate nella Figura 1.9.

Le informazioni proseguono per diverse pagine, giorno dopo giorno, fino al 1905. Dora Talbot pagava i lavoratori un dollaro al giorno e George B. Talbot (presumibilmente suo figlio) teneva le registrazioni. Alcuni lavoratori compaiono molte volte, altri soltanto una volta o due.

| Worked for Dora (1898) | |
|------------------------|--|
| Aug 6 | G B Talbot and team 1 day |
| Aug 6 | Dick Jones 1 day |
| Aug 6 | Albert Talbot 1 day |
| Aug 7 | G B Talbot and team 1 day |
| Aug 7 | Dick Jones 1 day |
| Aug 7 | Albert Talbot 1 day |
| Aug 9 | G B Talbot and team $\frac{1}{2}$ day in afternoon |
| Aug 9 | Albert $\frac{1}{2}$ day in afternoon |
| Aug 9 | Adah $\frac{1}{2}$ day in afternoon |
| Aug 10 | G B $\frac{1}{2}$ day in forenoon and 3 hours in afternoon |
| Aug 10 | Albert $\frac{1}{2}$ day in forenoon 3 hours afternoon |
| Aug 10 | Adah 3 hours in the afternoon |
| Aug 12 | G B $\frac{1}{2}$ day in forenoon 2 hours in afternoon |
| Aug 12 | Dick $\frac{1}{2}$ day in forenoon 2 hours in afternoon |
| Aug 12 | Albert $\frac{1}{2}$ day in forenoon 2 hours in afternoon |
| Aug 13 | G B and team 1 day |
| Aug 13 | Dick Jones 1 day |
| Aug 13 | Albert Talbot 1 day |
| Aug 13 | Adah Talbot $\frac{1}{2}$ day in the afternoon |
| Aug 14 | G B. and team 1 day |
| Aug 14 | Dick Jones 1 day |
| Aug 14 | Albert Talbot 1 day |
| Aug 14 | Adah Talbot $\frac{1}{2}$ day in afternoon |
| Aug 16 | G B. and team 1 day |
| Aug 16 | Dick Jones 1 day |

Figura 1.9 Il libro mastro di G.B. Talbot.

Qualche pagina è dedicata anche all'elenco di nomi e indirizzi dei lavoratori, come illustrato nella Figura 1.10. La relazione che collega le due tabelle è il nome dei lavoratori. Se alla fine del mese George avesse voluto mandare un incaricato a distribuire le buste paga a ogni lavoratore, che cosa avrebbe dovuto fare? Innanzitutto avrebbe dovuto sommare le cifre spettanti per ciascun lavoratore, mettere il denaro calcolato in ciascuna busta e scrivere il nome del lavoratore sulla stessa. Quindi avrebbe dovuto cercare ciascun indirizzo nella seconda parte del registro, scriverlo sulle buste e mandare l'incaricato a consegnarle.

G. B. Talbot possedeva un vero database relazionale, anche se utilizzava carta e inchiostro per raccogliere le informazioni, piuttosto che un'unità a disco di un computer. Anche se le tabelle sono poste in relazione grazie alle dita, agli occhi e alla mente di un uomo invece che con l'aiuto di una CPU, si tratta effettivamente di un database relazionale vero e proprio. È anche abbastanza assimilabile a quello che un progettista di applicazioni relazionali definirebbe *normalizzato*, termine che significa semplicemente che i dati sono raccolti in gruppi naturali: paga giornaliera e indirizzi non sono mescolati in un'unica sezione del registro.

Le voci di questo libro mastro, sia quelle riportanti le somme sia quelle degli indirizzi, potrebbero facilmente essere impostate come tabelle di ORACLE. Le domande o i compiti che G. B. Talbot doveva affrontare avrebbero potuto essere decisamente semplificati e nelle pagine seguenti viene spiegato come, utilizzando sia esempi moderni, sia le voci del vecchio registro di Talbot per scoprire le potenzialità di ORACLE.

| Addresses for Doc's Shop | |
|--------------------------------|---|
| Bart Jappon | Canner, Robot House, Hill St, Berkeley |
| Pat Lavery | Rock Hill Rd # 7, Easton |
| Lick Jones | Caprice Hotel, 9, Easton |
| * Arch Talbot | Boo King Rooming, 127 Main, 7, Easton |
| Andrew Sie | Pete Hill Rd # 3, 7, Easton |
| Hiram Wilson | Woolworth, Rock Hill |
| * Elbert Talbot | Woolback Rooming, 300 Main, 7, New Haven |
| Richard Koch and brother | Woolback Rooming, 7 |
| ? Bert Laverne | Corner Robot House, Hill St, Berkeley |
| Jed Hopkins | Matt Longfunk House, 3 Mile Rd, New Haven |
| Helen Brant | Pete Hill Rd, 7, Easton |
| William Living | Canner, Robot House, Hill St, Berkeley |
| George Docce | Pete Hill, of Pet Lavery |
| Elsie Hill | Matt Longfunk House, 3 Mile Rd, New Haven |
| Gerhard Lentzen | Boo King Rooming, 127 Main, Easton |
| Eliza Janisse | Solo Drury Farm, New Haven |
| Wilfred Lovell | ? |
| Roland French | Matt Longfunk House, 3 Mile Road, New Haven |
| Dorothy Lawson | (with Peter at Canner) |
| George Macauk and wife (Lily?) | with Wilf Longfunk |
| Alice Culpeper | of Elmer Edwards, New Haven |
| Dick Jones | Cor |
| Andrew Schuster | Hill 1, Ridge |

Figura 1.10 Gli indirizzi dei lavoratori nel libro mastro di G.B. Talbot.

• 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 **Perché vengono utilizzati i codici e non la lingua corrente?**
- 2.6 **Come arginare la confusione**
- 2.7 **Maiuscole e minuscole nei nomi e nei dati**
- 2.8 **Normalizzazione dei nomi**
- 2.9 **Cogliere l'opportunità**

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. In una tecnologia relativamente nuova come quella dei database relazionali, non è passato abbastanza tempo per consentire alla maggior parte delle aziende di avere sufficiente esperienza su che cosa evitare e come.

Se 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, il pericolo aumenta.

In questo capitolo vengono discussi alcuni dei pericoli che progettisti e utenti dovrebbero considerare. Nella Parte quarta vengono trattati questi e altri argomenti con maggiore dettaglio, in particolare quelli che interessano i progettisti nel loro compito di sviluppo di un'applicazione efficace e produttiva.

2.1 **È davvero facile come sembra?**

Secondo i fornitori di database, sviluppare un'applicazione utilizzando un database relazionale e gli strumenti correlati di “quarta generazione” sarà addirittura 20 volte più rapido che utilizzare la progettazione di sistema tradizionale.

È molto facile: in ultima analisi, i programmatori e gli analisti di sistema non risulteranno più indispensabili poiché gli utenti finali controlleranno appieno i loro destini.

I critici dell'approccio relazionale tuttavia, ritengono che i sistemi relazionali siano per natura più lenti degli altri, che gli utenti a 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 relazionale di quarta generazione presuppone richieste molto diverse nei confronti delle società e della gestione rispetto ai modelli di sviluppo tradizionale. 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 tutto 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 e ingenuità 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 si rivelano 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 vecchi sono generalmente più lenti. Vengono imposti controlli di progetto per assicurare una revisione generale e la qualità del prodotto, ma fondamentalmente i compiti dei metodi più vecchi, codifica, compilazione, linking e collaudo, vengono effettuati con un ritmo più lento. Il ciclo, soprattutto per un mainframe, è spesso così tedioso che i programmatori impiegano molto tempo a “controllare a tavolino” il prodotto onde evitare il rallentamento prodotto da 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, scompaiono anche i controlli a tavolino. L'inclinazione di molte persone 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 veloce aggiornamento; se il prodotto non è abbastanza veloce, si può aggiustare mentre si utilizza: l'importante è portarlo a compimento prima della scadenza fissata 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. Hanno imparato la teoria e la progettazione relazionale od orientata agli 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 di ORACLE dovrebbe essere più lungo e più completo rispetto a quello di un progetto tradizionale. Ciò è vero anche quando vengono impostati controlli di progetto appropriati e anche se il progetto è guidato da manager esperti, poiché ci sono meno procedure di verifica a tavolino e una sopravvalutazione innata. Con questi collaudi deve essere controllata la correttezza delle schermate e dei report, del caricamento e dell'aggiornamento, 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

Molte persone attendono con ansia il giorno in cui si potrà dire semplicemente, come il Capitano Kirk "Computer..."; esporre domande in linguaggio corrente e ottenere la risposta, visualizzata sullo schermo del computer, in pochi secondi.

Siamo molto più vicini a questi obiettivi 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 possibile concepire facilmente sistemi basati sulla lingua naturale che siano di comprensione e utilizzo immediato per gli utenti non esperti. Il potenziale è questo, già disponibile con il database e gli strumenti di ORACLE, ma soltanto poche persone lo hanno compreso e sono in grado di sfruttarlo.

La chiarezza e la comprensibilità dovrebbero essere i tratti distintivi di qualsiasi applicazione ORACLE. Si può utilizzare la lingua corrente, gli utenti finali senza preparazione specifica nella programmazione possono utilizzare facilmente le applicazioni e avere informazioni formulando una semplice query.

E come si fa a ottenere tutto ciò? Innanzitutto, uno degli obiettivi fondamentali durante la programmazione deve essere quello di rendere l'applicazione facile da capire e semplice da utilizzare. Anche se si sbaglia, questa è la direzione da seguire, perfino 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

Nel 1969 il costo per tenere in attività un computer con una velocità di elaborazione di quattro milioni di istruzioni per secondo (MIPS) era di circa mille dollari l'ora. Per il 1989, il costo era sceso a 45 dollari l'ora e ha continuato a precipitare fino a oggi. 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 nella 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, poi su Macintosh, ora con X-Windows, MS-Windows, Presentation Manager, New Wave, NeXT e altri sistemi di impostazione 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

Questo problema si verifica soprattutto per i codici, le abbreviazioni e gli standard di denominazione, che vengono completamente ignorati quando sono considerate 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.

Una volta le registrazioni economiche venivano effettuate su libri mastri e giornali. Ciascun evento e transazione veniva trascritto, riga dopo riga, utilizzando la lingua corrente. Si osservi il registro di Talbot, nella Figura 2.1: ci sono forse dei codici? Assolutamente no. Delle abbreviazioni? Sì, qualcuna di uso comune, immediatamente comprensibile da qualsiasi lettore di lingua inglese. Quando Talbot vendette una corda di legna il 28 gennaio, annotò: "Jan 28 (1 Crd) Wood Methest Church 2.00".

In molte applicazioni odierne, la stessa transazione sarebbe rappresentata nei file di un computer da un numero come "028 04 1 4 60227 3137"; la data secondo il calendario giuliano per indicare il ventottesimo giorno dell'anno, il codice di transazione 04 (una vendita) la quantità 1 del tipo di quantità 4 (una corda) dell'articolo

| | | | 15 |
|--------|--------------------------------------|------|------|
| Jan 15 | Telephone to Coopertown | | |
| Jan 20 | For team in barn | 50 | |
| Jan 20 | team to drive to Coopertown | 1 50 | |
| Jan 20 | team in barn to : : : | 50 | |
| Jan 22 | (for sugar bread and molasses) Jan 3 | 1 05 | |
| Jan 24 | 1/2 lb of tea | 50 | 25 - |
| Jan 25 | had for Shirley | 10 | |
| Jan 27 | Received of R H Boutwell | | 7 50 |
| Jan 28 | for Salt water | 8 | |
| Jan 28 | 6 envelopes | 12 | |
| Jan 28 | 2 Quarts of Creoal | 75 | |
| Jan 28 | 1/1 Crd Wood Methest Church | | 2 00 |
| Jan 28 | Shirley's Schooling | 1 00 | |
| Jan 30 | 186 Tommey | 120 | 2 23 |
| Jan 30 | 496 Corn | 60 | 4 45 |
| Jan 31 | for grinding | | 90 |
| Jan 31 | for dimond Eyes and braid | | 20 |
| Jan 31 | pop Corn 5lb @ .- | | 20 |
| Feb 2 | for dimond Eyes | | 20 |
| Feb 2 | for Shirley Shoes Methest | | 50 |
| Feb 14 | 102. Tommey | 120 | 1 22 |
| Feb 14 | 5 lb of sulphur | 5 | 25 |
| Feb 17 | 1/2 sulphur | | 11 |
| Feb 17 | 4 gal K oil | | 32 |
| Feb 17 | fouler solution | | 25 |

Figura 2.1 Una pagina del libro mastro di Talbot.

60227 (60=legno, 22=non finito, 7>tagliato) al cliente 3137 (Chiesa Metodista). Gli addetti all'inserimento di questi dati dovrebbero in effetti conoscere o cercare la gran parte di questi codici e digitarli 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 main-frame 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 e IMS. La potenzialità e le caratteristiche di un database relazionale sono virtualmente sprecate, quando vengono utilizzate in questo modo.

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

Perché utilizzare i codici? Sono due le giustificazioni che vengono 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 nel computer.

Il secondo punto è un anacronismo, un prodotto dell'epoca in cui quattro MIPS costavano 1 dollaro all'ora (o anche di più). La memoria di sistema (un tempo ferrite a forma di frittella montata su griglie metalliche) e la memoria permanente (nastri o grossi piatti magnetici) erano talmente costose e le CPU talmente lente (con meno potenza di una calcolatrice tascabile) 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 (come 3137 per Kentgen, Gerhardt) riducono ancora di più le prestazioni richieste alla macchina.

Poiché 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, che dovettero 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 agli uomini e funzionare con il linguaggio umano con parole che gli umani comprendono. Era ora che accadesse! Eppure, senza porsi il problema consapevolmente, sviluppatori e progettisti continuano volentieri o nolenti a utilizzare i codici, come se fosse ancora il 1969.

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 i numeri

“3137” piuttosto che “Kentgen, Gerhardt”. Questa giustificazione non è vera nel caso di ORACLE. Non soltanto è più costoso addestrare le persone a conoscere il codice corretto per clienti, prodotti, transazioni e altro, oltre al 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 “Kentgen, Gerhardt” 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, virtualmente inesplorate, 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 in conti sospesi 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 a 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. Per gli utenti che sono stati sommersi da manuali sui codici, lavorare con un’applicazione basata sulla lingua comune produce un effetto altamente rilassante.

2.6 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”, ad 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. normalizzare i dati;
2. selezionare le denominazioni per le tabelle e le colonne;
3. selezionare le 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 denominati 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 di 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 a 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. Questo argomento viene trattato in maniera più completa nella Parte quarta, in particolare per quanto concerne gli sviluppatori.

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. G.B. Talbot teneva traccia delle diverse persone che lavoravano per lui. La maggior parte svolgeva un lavoro occasionale e abitava in uno dei molti alloggi della città. Talbot studiò una forma semplice per raccogliere queste informazioni, come illustrato nella Figura 2.2.

| | |
|----------|--|
| Name: | John Pearson Age 27 |
| Lodging: | Rox Hill for Men |
| Man: | John Peletier |
| Address: | N. Compton |
| Skills: | <ul style="list-style-type: none"> 1. combine driver, harness, drive, groom horses, adjust blades. 2. average smithy, can stack for fire, run bellows, cut shoe horses. 3. good woodcutter, can mark and fell trees, split, stack, haul. ? ? |

Figura 2.2 Informazioni sul lavoratore.

Seguendo tecniche più antiche, Talbot avrebbe potuto facilmente sviluppare un database che si adattasse alla disposizione fondamentale di questa forma. Sembra un approccio piuttosto rigido, che segue fondamentalmente il modello a schedario cartaceo. Molte applicazioni sono state progettate nella stessa maniera. I progettisti prendono le copie dei moduli esistenti (fatture, ricevute di vendita, domande di lavoro, dati di fatto, tabulati di informazioni sui lavoratori) e creano dei sistemi sulla base del loro contenuto e della loro forma.

Se ci si pensa bene, tuttavia, ci sono alcuni problemi nascosti. Si supponga che lo schema di Talbot diventi la struttura di una tabella in ORACLE. La tabella potrebbe essere denominata LAVORATORE e le colonne potrebbero essere Nome, Alloggio, Direttore, Indirizzo, Compito1, Compito2 e Compito3. Si osservi la Figura 2.3. Gli utenti di questa tabella hanno già un problema: sui fogli di carta Talbot poteva elencare tutti i compiti desiderati, mentre nella tabella LAVORATORE gli utenti devono limitarsi a tre compiti.

Si supponga anche che, oltre al metodo dello schedario cartaceo, Talbot digitò gli stessi elementi di informazione in ORACLE per verificare la sua tabella di database LAVORATORE. Ogni scheda di carta diventa una riga di informazioni. Ma che cosa accade quando Peletier (il direttore del registro di Talbot) si trasferisce nel New Hampshire e un nuovo direttore prende il suo posto? Occorre scorrere il modulo di ciascun lavoratore (nello schedario cartaceo, ovvero ogni riga della tabella LAVORATORE) e correggere tutte le volte che compare il nome di Peletier.

Tabella LAVORATORE

Nome
Eta
Alloggio
Direttore
Indirizzo
Compito1
Compito2
Compito3
.
.
?
.

Figura 2.3 La tabella LAVORATORE di Talbot con problemi di mancanza di dati.

E quando Rose Hill (la residenza dove vive Pearson) viene acquistata da Major Resorts International, l'indirizzo cambia in “Residenza Major Resorts” e di nuovo tutte le registrazioni dei lavoratori devono essere modificate. Che cosa farà Talbot quando John Pearson aggiungerà un quarto compito all'elenco? E “buono” o “medio” fanno effettivamente parte del compito, oppure rappresentano un livello di capacità che forse dovrebbe costituire una colonna a parte?

I problemi appena elencati non sono questioni legate all'uso del computer o tecniche, anche se vengono evidenziate durante la progettazione di un database. Si tratta soprattutto di questioni di base sulle modalità di organizzare in modo “sensibile” 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.

Prima forma normale Nella prima fase occorre impostare i dati nella prima forma normale. 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 accade per i compiti nel modulo cartaceo di Talbot (che viene ridotto con soltanto tre compiti nella prima elaborazione della tabella LAVORATORE).

Invece di mantenere tre compiti per lavoratore, i compiti di ciascuno sono collocati in una tabella separata, con una riga per nome, compito e descrizione di quest'ultimo. In questo modo viene eliminata la necessità di un numero variabile di compiti nella tabella LAVORATORE (in ogni caso impossibile per ORACLE) e si procede in modo più funzionale rispetto a limitare la tabella LAVORATORE a tre soli compiti.

Successivamente occorre definire la chiave primaria di ciascuna tabella: quale elemento può identificare una tabella in maniera univoca e consentire di estrarlar-

ne una riga di informazioni? Per semplicità, si presume che i nomi dei lavoratori siano unici, quindi Nome è la chiave primaria per la tabella LAVORATORE. Poiché ogni lavoratore può avere diverse righe nella tabella COMPITO, Nome più Compito è la chiave primaria completa per la tabella COMPITO (due parti vengono combinate per formare un intero). Si osservi in proposito la Figura 2.4.

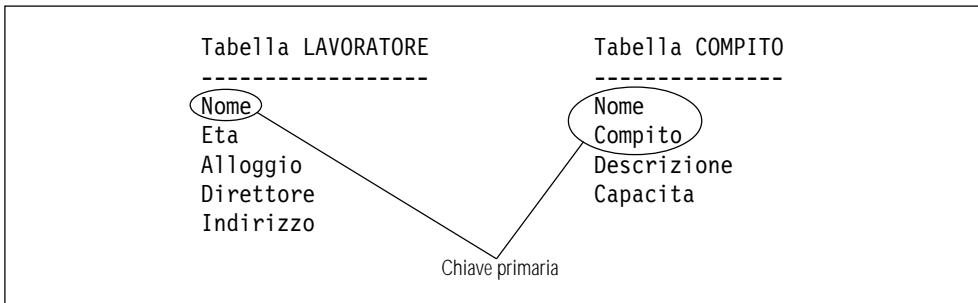


Figura 2.4 I dati sui lavoratori di Talbot in prima forma normale.

Per scoprire il grado di capacità di John Pearson come taglialegna e per ottenere la descrizione dei compiti di un taglialegna è sufficiente digitare la seguente query:

```
select Capacita, Descrizione
      from COMPITO
```

La risposta di ORACLE è:

```
Capacita Descrizione
-----
Buona Segnare e abbattere alberi, Spaccare, Accatastare, Trasportare
```

Quale elemento può condurre a una sola riga nella tabella COMPITO? Sia Nome, sia Compito. Ma Descrizione dipende soltanto da Compito, indipendentemente dal nome inserito. A questo punto si passa alla fase successiva.

Seconda forma normale Nella fase due, la seconda forma normale, è richiesta la selezione dei dati che sono dipendenti soltanto da una parte della chiave. Per impostare la seconda forma normale, occorre trasferire Compito e Descrizione su una terza tabella. La chiave primaria per la terza tabella è soltanto Compito e la sua ampia descrizione appare una sola volta. Se la tabella viene lasciata nella prima forma normale, le lunghe descrizioni sarebbero ripetute per ogni lavoratore con quel particolare compito. Inoltre, se l'ultimo fabbro lasciasse la città, con l'eliminazione del suo nome dal database verrebbe eliminata anche la descrizione del compito. Con la seconda forma normale, compito e descrizione possono essere conservati nel database anche se al momento nessun lavoratore li possiede. I compiti possono anche essere aggiunti, come “descrizione del lavoro”, prima di collocare nella tabella la persona a cui sono assegnati. Si osservi a questo proposito la Figura 2.5.

| Tabella LAVORATORE | Tabella COMPITOLAVORATORE | Tabella COMPITO |
|--------------------|---------------------------|-----------------|
| Nome | Nome | Compito |
| Eta | Compito | Descrizione |
| Alloggio | Capacita | |
| Direttore | | |
| Indirizzo | | |

Figura 2.5 I dati sui lavoratori di Talbot in seconda forma normale.

Terza forma normale Nella fase tre, la terza forma normale, occorre sbarazzarsi di tutti gli elementi delle tabelle che non sono dipendenti unicamente alla chiave primaria. Le informazioni relative all'alloggio del lavoratore sono dipendenti dal fatto che il lavoratore risiede in un dato luogo (se si trasferisce, occorre aggiornare la riga corrispondente con il nome della nuova abitazione), ma il nome del direttore dell'alloggio e l'indirizzo non sono dipendenti dal fatto che il lavoratore vi abiti.

Le informazioni relative all'alloggio vengono quindi trasferite in una tabella separata e, per motivi di opportunità, viene utilizzata una versione abbreviata della denominazione dell'alloggio come chiave primaria e il nome completo viene mantenuto come NomeLungo. Nella Figura 2.6 sono illustrate le tabelle nella terza forma normale e nella Figura 2.7 viene rappresentata graficamente la relazione tra le tabelle stesse.

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.

Occorre semplicemente 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”.

| LAVORATORE | COMPITOLAVORATORE | COMPITO | ALLOGGIO |
|------------|-------------------|-------------|-----------|
| Nome | Nome | Compito | Alloggio |
| Eta | Compito | Descrizione | NomeLungo |
| Alloggio | Capacita | | Direttore |
| | | | Indirizzo |

Figura 2.6 I dati sui lavoratori di Talbot in terza forma normale.

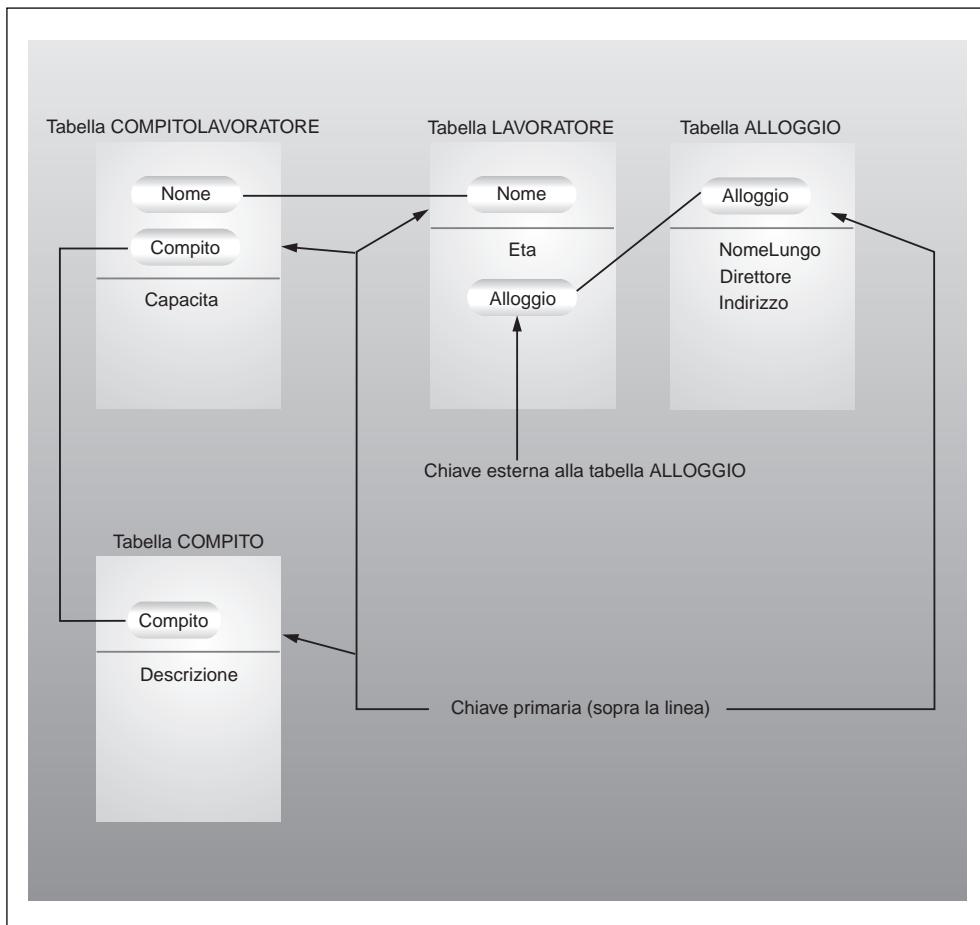


Figura 2.7 Relazioni tra le tabelle dei lavoratori.

Navigare tra i dati Il database di Talbot è ora nella terza forma normale. Nella Figura 2.8 viene presentato un campione di quello che potrebbero contenere queste tabelle. La correlazione tra le quattro tabelle risulta immediatamente evidente. Si può navigare da una tabella all'altra per estrarre informazioni su un determinato lavoratore, sulla base delle chiavi impostate per ciascuna tabella. In ogni tabella, con la chiave primaria è possibile identificare in maniera univoca una singola riga. Ad esempio, se si seleziona John Pearson, si può facilmente scoprirne l'età nella tabella LAVORATORE, perché Nome è la chiave primaria di questa tabella.

Cercando il nome e il compito “Taglialegna” nella tabella COMPITOLAVORATORE si può trovare il grado di capacità, “Buona”.

Nome e Compito insieme sono le chiavi primarie di questa tabella, mediante le quali si seleziona unicamente una singola riga. Se si cerca la voce “Taglialegna” nella tabella COMPITO si può leggere la descrizione completa delle capacità richieste per questa qualifica.

| Tabella LAVORATORE | | | |
|--------------------|-----|------------|--|
| NOME | ETA | ALLOGGIO | |
| Adah Talbot | 23 | Papa King | |
| Bert Sarjeant | 22 | Crammer | |
| Dick Jones | 18 | Rose Hill | |
| Elbert Talbot | 43 | Weitbrocht | |
| Helen Brandt | 15 | | |
| Jed Hopkins | 33 | Matts | |
| John Pearson | 27 | Rose Hill | |
| Victoria Lynn | 32 | Mullers | |
| Wilfred Lowell | 67 | | |

| Tabella COMPITOLAVORATORE | | |
|---------------------------|-------------|--------------|
| NOME | COMPITO | CAPACITA |
| Adah Talbot | Operaio | Buono |
| Dick Jones | Fabbro | Eccellente |
| Elbert Talbot | Aratore | Lento |
| Helen Brandt | Trebbiatore | Molto veloce |
| John Pearson | Trebbiatore | |
| John Pearson | Taglialegna | Buono |
| John Pearson | Fabbro | Medio |
| Victoria Lynn | Fabbro | Preciso |
| Wilfred Lowell | Operaio | Medio |
| Wilfred Lowell | Aratore | Medio |

| Tabella COMPITO | |
|-----------------|--|
| COMPITO | DESCRIZIONE |
| Trebbiatore | Badare ai cavalli, condurli, regolare le lame |
| Aratore | Badare ai cavalli, condurli, arare |
| Scavatore | Segnare e aprire il terreno, scavare, puntellare, riempire |
| Fabbro | Accendere fuoco, usare soffietto, tagliare, ferrare cavalli |
| Taglialegna | Segnare e abbattere alberi, spaccare, accatastare, trasportare |
| Operaio | Lavoro generico senza compiti particolari |

| Tabella ALLOGGIO | | | |
|------------------|-----------------------|---------------|--------------------|
| ALLOGGIO | NOMELUNGO | DIRETTORE | INDIRIZZO |
| Cranmer | Cranmer Retreat House | Thom Cranmer | Hill St, Berkeley |
| Matts | Matts Long Bunk House | Roland Brandt | 3 Mile Rd, Keene |
| Mullers | Mullers Coed Lodging | Ken Muller | 120 Main, Edmeston |
| Papa King | Papa King Rooming | William King | 127 Main, Edmeston |
| Rose Hill | Rose Hill For Men | John Peletier | Rfd 3, N. Edmeston |
| Weitbrocht | Weitbrockt Rooming | Eunice Benson | 320 Geneva, Keene |

Figura 2.8 Informazioni nelle tabelle di Talbot.

Quando è stato cercato il nome John Pearson nella tabella LAVORATORE, sono stati visualizzati anche i dati relativi all'alloggio, ovvero Rose Hill. Questo dato è la chiave primaria della tabella ALLOGGIO. Se si cerca la voce Rose Hill in questa tabella, si trova il nome completo “Rose Hill For Men”, il direttore “John Peletier”, e l'indirizzo “Rfd 3, N. Edmeston”. Quando la chiave primaria di una tabella appare in un'altra tabella, come accade in questo caso con le tabelle ALLOGGIO e LAVORATORE, viene definita *chiave esterna*.

Osservando queste tabelle si possono anche trarre conclusioni sul mondo reale: Talbot non sa dove vivono Brandt o Lowell, non riporta nelle sue registrazioni la qualifica di Sarjeant o Hopkins e non accerta il grado di capacità di Pearson come trebbiatore. Poiché i dati sono organizzati in maniera logica, Talbot tiene una registrazione per il compito di scavatore, anche se nessuno dei lavoratori attuali lo esercita.

Si tratta di una forma di organizzazione dell'informazione di tipo logico e “sensibile”, 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.

Ad esempio, si potrebbe spezzare la voce dell'indirizzo nei suoi componenti, come “via”, “città”, “provincia” e così via. Nome potrebbe essere suddiviso in “Nome” e “Cognome”, e si potrebbero restringere le opzioni della colonna Capacità nella tabella COMPITOLAVORATORE.

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 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 i tecnici nella Parte 4.

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 a cui viene data troppo poca importanza, anche degli esperti.

Le denominazioni di tabella e di colonna vengono spesso scelte senza consultare gli utenti finali e senza una revisione rigorosa durante il processo di progettazione. Entrambe queste mancanze possono avere serie conseguenze quando viene il momento di utilizzate effettivamente l'applicazione.

Ad esempio, si considerino le tabelle di Talbot riportate nel precedente paragrafo, che contengono le seguenti colonne:

| LAVORATORE | COMPITOLAVORATORE | COMPITO | ALLOGGIO |
|------------|-------------------|-------------|-----------|
| Nome | Nome | Compito | Alloggio |
| Eta | Compito | Descrizione | NomeLungo |
| Alloggio | Capacita | | Direttore |

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 Nome, Eta, Alloggio
  from LAVORATORE
 order by Eta;
```

Gli utenti sono in grado di comprendere 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 Talbot avesse scelto i nomi seguenti?

| Tabella | LAVORATORI | TL | COMPITO | ACCOMODAMENTO |
|---------|------------|-----------|---------|---------------|
| nomelav | | nomelavtl | comp | nameaccom |
| etalav | | comptl | des | namecomplet |
| nomeall | | capac | | dir ind |

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. Nella Tabella 2.1 sono mostrati alcuni esempi reali di denominazioni di colonna e tabella tratte proprio da queste fonti.

Tabella 2.1 Nomi di tabelle e colonne di varia provenienza.

| TABELLE | COLONNE | | |
|-----------|-------------|-----------|------------|
| REP | ADD1 | NOMIMP | NOTE |
| IMP | COGN | NOMEIMP | NUM_ORD |
| IMPI | ORD | NUMIMP | NOMI |
| MIEIIIMP | BLOC | VENDIMP | NI |
| PE | CDLEXP | ALTAQTA | DATAPUB |
| PERSONALE | NUMREP | ALTORANG | OTA |
| PROG | TIPOSCONTTO | BASSORANG | PCTTASVEND |
| TITOLI | RNOME | BASQUANT | LAVRTR |

Nel seguito sono descritti 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 essere allo stesso modo dei codici, poiché in entrambi i casi occorre decifrarli.
- *Abbreviazioni incongruenti.* In un caso si utilizza “BASSO”, in un altro “BAS”. Qual è la dicitura utilizzata, “NUMERO”, “NUM” oppure “N”? “NUMIMP” oppure “NI”? “NOMEIMP” oppure “NOMIMP”?
- *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 “PE”? “PCTTASVEN”? “CDLEXP”?
- *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?
- *Utilizzo incongruente del plurale.* “NOME” oppure “NOMI”?
- *Le regole apparentemente impiegate hanno dei limiti evidenti.* Se la prima lettera della denominazione di tabella viene utilizzata come nome di colonna, come accade in “RNOME”, che cosa succede quando è necessario utilizzare una tabella denominata “REPARTO”? Anche in questo caso viene assegnato un nome di colonna come “RNOME”? Ma allora, perché in entrambi i casi la colonna in questione non viene semplicemente denominata “NOME”?

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 LAVORATORE e in questo modo verranno notevolmente minate l'accettazione e l'utilità dell'applicazione.

In passato era richiesto che i nomi di campo e di file fossero lunghi al massimo sei od otto caratteri. Come risultato, i nomi erano inevitabilmente un miscuglio confuso di lettere, numeri e abbreviazioni criptiche. Come molte altre limitazioni della tecnologia precedente, questa non è più necessaria. 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. Queste convenzioni vengono trattate più ampiamente nel Capitolo 35. 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.

Nomi in lingua corrente per i dati

Dopo aver sollevato l'importante questione delle convenzioni di denominazione, il passo successivo è quello di 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. Seguendo ancora l'esempio di Talbot, la categoria Compito avrebbe potuto essere impostata con un codice a due cifre, in cui 01 significa fabbro, 02 trebbiatore e così via, e 99 "lavoro generico senza compiti particolari". Il grado di capacità avrebbe potuto essere impostato secondo una scala da 1 a 10. Si ottiene un miglioramento? Se si chiede a una terza persona informazioni su Dick Jones, è accettabile la risposta "9 per 01"? Perché dovrebbe essere permesso a una macchina di essere meno chiara, in particolare quando è semplice progettare applicazioni che possano comunicare l'informazione "fabbro eccellente"?

Inoltre, avere informazioni in lingua corrente rende più facile impostare e capire le query. Quale delle due domande e risposte in SQL riportate di seguito risulta più ovvia? La seguente:

```
select nomelav, capac, comptl
      from t1;
```

| NOMELAV | CA TL |
|----------------|--------|
| ----- | --- -- |
| Adah Talbot | 07 99 |
| Dick Jones | 10 01 |
| Elbert Talbot | 03 03 |
| Helen Brandt | 08 02 |
| John Pearson | 02 |
| John Pearson | 07 04 |
| John Pearson | 05 01 |
| Victoria Lynn | 09 01 |
| Wilfred Lowell | 05 99 |
| Wilfred Lowell | 05 03 |

o quest'altra:

```
select Nome, Capacita, Compito
      from COMPITOLAVORATORE;
```

| NOME | CAPACITA | COMPITO |
|----------------|--------------|-------------|
| ----- | ----- | ----- |
| Adah Talbot | Buono | Operaio |
| Dick Jones | Eccellente | Fabbro |
| Elbert Talbot | Lento | Aratore |
| Helen Brandt | Molto veloce | Trebbiatore |
| John Pearson | | Trebbiatore |
| John Pearson | Buono | Taglialegna |
| John Pearson | Medio | Fabbro |
| Victoria Lynn | Preciso | Fabbro |
| Wilfred Lowell | Medio | Operaio |
| Wilfred Lowell | Medio | Aratore |

2.7 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.

Questa caratteristica viene contrabbadata come un vantaggio poiché, ad esempio, permette ai programmatore di creare molte tabelle con nomi simili. Si può creare una tabella lavoratore, una Lavoratore, una LAVORAtore e così via, all'infinito. Sono tutte tabelle diverse. Come si può supporre che chiunque, compreso il programmatore, possa 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. Utilizzando determinati comandi, sono possibili modalità di ricerca delle informazioni da 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 rendono possibile 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.

2.8 Normalizzazione dei nomi

Esistono molti prodotti sul mercato 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 come quelli presentati nella Tabella 2.1. 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 risolta semplicemente facendo subito un buon lavoro di impostazione iniziale?

Per motivi legati al miglioramento delle prestazioni può accadere che alcuni dei dati di un'applicazione siano 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 estrappolazione, 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 desiderano creare nuovi report, o report ad hoc, la loro possibilità di farlo in maniera veloce e accurata viene 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.

2.9 Cogliere l'opportunità

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 pretenderla. Entrambe le categorie ne trarrebbero enormi benefici.

- ## Parte seconda

SQL: da principianti a esperti

•
•
•
•
• Capitolo 3

• **Le parti fondamentali del discorso in SQL**

- •
• 3.1 **Stile**
• 3.2 **Utilizzo di SQL per selezionare dati
da tabelle**
• 3.3 **I comandi select, from, where e order by**
3.4 **Logica e valore**
3.5 **LIKE**
3.6 **Un altro impiego delle sottoquery
con where**
3.7 **Come combinare le tabelle**
3.8 **Creazione di una vista**

Con SQL (Structured Query Language), si può 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.

Nel Capitolo 1 è stato spiegato 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. Viene inoltre 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 impartite.

Con questo strumento 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; ad 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 stessa tabelle proposte in questo testo e ci si esercita impostando le stesse query.

Nell'Appendice A sono contenute le istruzioni per caricare tali tabelle. Presupponendo che ciò sia già stato fatto, occorre connettersi a SQL*PLUS e iniziare a lavorare digitando:

```
sqlplus
```

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. Da questo punto in poi, si fa riferimento a SQLPLUS senza l'asterisco). Poiché in ORACLE sono previsti accurati strumenti di protezione dei dati memorizzati, ogni volta che ci si connette viene richiesto un ID e una password. Viene visualizzato un messaggio di copyright e quindi la richiesta di nome utente e password. Per accedere alle tabelle descritte in questo testo, occorre digitare la parola esercizi sia come nome utente, sia come password. 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 si utilizza un PC e viene visualizzato questo messaggio:

Comando o nome di file errato.

cio può significare tre cose: la sottodirectory non è quella corretta, ORACLE non è nel percorso indicato, oppure non è stato installato in maniera appropriata. Su altri tipi di computer, messaggi analoghi indicano che ORACLE non è pronto per essere avviato. Se si osserva questo messaggio:

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

può significare che è stato inserito in maniera errata il nome utente o la password, oppure che il nome utente esercizi 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 oppure seguire le indicazioni di installazione riportate nell'Appendice A. 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**, basta 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** vengono verificati i termini di un'uguaglianza nel database.

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 **Sezione** deve essere verificata la presenza della lettera '**F**' e non ricercata una colonna denominata **F**. L'apice singolo su un lato della lettera indica 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 e i listati più agevoli 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**, **SOUNDEX** 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 normale: **GIORNALE**, **CLIMA**, **LOCAZIONE** 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; o ancora, si può 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 Utilizzo di SQL per selezionare dati da tabelle

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

```
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 |
| Film | B | 4 |
| Bridge | B | 2 |
| Necrologi | F | 6 |

14 rows selected.

Che cosa c'è di diverso tra la tabella riportata in precedenza e quella tratta dal giornale della Figura 3.1? In entrambe sono riportate le stesse informazioni, ma il formato cambia; ad 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 riga 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 possibile modificare tali parametri, ma 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 viene mostrato nel Capitolo 6).

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.

| 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 di sezioni di giornale.

Occorre semplicemente 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.

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. Creando questa tabella con il parametro NOT NULL per la colonna Argomento, sono state fornite istruzioni a ORACLE in modo da non consentire di inserire informazioni che non fossero effettivamente voci della colonna Argomento. Non è possibile inserire una nuova riga in questa tabella lasciando vuota la colonna Argomento (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 il vincolo specifico di non poter essere vuota.

Poiché Sezione e Pagina non hanno 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 si possono 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 della tabella GIORNALE viene visualizzata con ampiezza pari a dieci caratteri, anche se non ci sono pagine il cui numero ha più di due cifre.

Ciò accade perché 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 dei titoli di colonna impostato da SQLPLUS. È possibile modificare queste caratteristiche a seconda delle necessità, come 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 i messaggi di feedback utilizzando il comando feedback, come illustrato di seguito:

```
set feedback off
```

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

```
set feedback 25
```

Con quest'ultimo comando si indica a ORACLE che non si desidera conoscere il numero di righe visualizzate se questo numero non è di almeno 25 righe.

A meno che non si imposti un parametro diverso, la funzione di feedback è impostata a 6 righe.

`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 nei capitoli seguenti. Per un elenco completo, si può consultare la voce `set` nel Capitolo 37.

Il comando `set` ha una controparte nel comando `show`, con il quale vengono visualizzate le istruzioni impostate. Ad 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 10
```

Poiché il valore 10 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
```

Tuttavia, ciò 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 viene trattato nel Capitolo 5.

3.3 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, `from` indica i nomi delle tabelle in cui si trovano tali colonne. Riprendendo le procedure illustrate in precedenza con la tabella **GIORNALE**, si può osservare come sono stati utilizzati questi comandi. Nella prima riga 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 in SQLPLUS solitamente termina con il segno di due punti (talvolta definito *terminatore SQL*). La parola chiave `where` indica a ORACLE quali qualificatori inserire nelle informazioni selezionate.

Ad 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 selezionate siano visualizzate in un determinato ordine, si utilizza il comando `order by`. Si possono impostare vari gradi di selezione dell’ordine; si considerino gli esempi seguenti:

```
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 utilizzando order by può essere invertito 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 e 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 definite *clausole*. Nella Figura 3.2 sono illustrati alcuni esempi di ciascun tipo di clausola.

3.4 Logica e valore

Anche la clausola con where, proprio come quella con order by, può essere composta da più parti, ma con un grado di raffinamento decisamente superiore. Il grado di utilizzo di where viene controllato mediante l'uso attento di istruzioni logiche impartite a ORACLE secondo le esigenze del momento. Le istruzioni vengono espresse utilizzando simboli matematici detti *operatori logici*. Gli operatori logici vengono elencati con una breve spiegazione nel Capitolo 37 singolarmente per nome e anche raggruppati sotto l'intestazione “Operatori logici”.

| | |
|---|---|
| <pre>select Argomento, Sezione, Pagina from GIORNALE where Sezione = 'F'</pre> | <i><--clausola select</i> <i><--clausola from</i> <i><--clausola where</i> |
|---|---|

Figura 3.2 Clausole relazionali.

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

| ARGOMENTO | S | PAGINA |
|-----------|---|--------|
| Necrologi | F | 6 |
| Salute | F | 6 |

Ecco un semplice esempio di relazione tra logica e valore, in cui i valori della colonna Pagina vengono visualizzati solo se uguali a 6. Viene visualizzata ogni riga per la quale l'uguaglianza risulta vera.

Qualsiasi riga in cui Pagina non è uguale a 6 (in altre parole, le righe in cui l'uguaglianza Pagina = 6 è falsa) viene saltata.

Il segno di uguale viene definito come operatore logico, perché lo si utilizza effettuando una verifica logica, ovvero confrontando i valori posti alla sua sinistra e alla sua destra di esso, 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 deve essere verificato (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

Si può utilizzare un singolo operatore logico per verificare un singolo valore, come viene spiegato nel paragrafo “Test logici su un singolo valore”. Si osservi qualche esempio tra quelli elencati: funzionano tutti in maniera analoga e possono essere combinati a seconda delle esigenze, anche se devono essere seguite delle regole specifiche sulle modalità di azione reciproca.

Uguale, maggiore, minore, diverso Con i test logici si possono confrontare dei valori, sia con il parametro di uguaglianza, sia con parametri relativi. Di seguito viene presentato 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 |

Occorre prestare particolare attenzione quando si utilizzano gli operatori “maggiore 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’.

Test logici su un singolo valore

Tutti 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 |

Poiché su alcune tastiere manca il punto esclamativo o l’accento circonflesso (^), ORACLE prevede tre modi per specificare l’operatore di disegualanza. L’alternativa finale, <>, va considerata come operatore di disegualanza 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 ‘__I%’ | Argomento ha una ‘I’ nella terza posizione |
| Argomento LIKE ‘%0%0%’ | Argomento contiene due ‘o’. |

IS NULL, IS NOT NULL

| | |
|----------------------------|---------------------------|
| Precipitazione IS NULL | “Precipitazione è ignoto” |
| Precipitazione IS NOT NULL | “Precipitazione è noto” |

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



3.5 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 modello descritto. Vendono utilizzati due caratteri speciali per contrassegnare il tipo di verifica che si desidera 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 'M%';
```

| 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 segni di percentuale multipli. 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%o%';
```

| ARGOMENTO | S | PAGINA |
|-----------|---|--------|
| Necrologi | F | 6 |

Si confronti 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%i%';
```

| 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 tabella COMFORT che segue contiene, oltre ad altri dati, le precipitazioni rilevate a San Francisco e Keene, nel New Hampshire (Stati Uniti), per quattro date campione durante l’anno 1993.

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

| CITTA | DATA CAMPIONE | PRECIPITAZIONE |
|---------------|---------------|----------------|
| SAN FRANCISCO | 21-MAR-93 | .5 |
| SAN FRANCISCO | 22-JUN-93 | .1 |
| SAN FRANCISCO | 23-SEP-93 | .1 |
| SAN FRANCISCO | 22-DEC-93 | 2.3 |
| KEENE | 21-MAR-93 | 4.4 |
| KEENE | 22-JUN-93 | 1.3 |
| KEENE | 23-SEP-93 | |
| KEENE | 22-DEC-93 | 3.9 |

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

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

| CITTA | DATAACAMPPI | PRECIPITAZIONE |
|-------|-------------|----------------|
| KEENE | | 23-SEP-93 |

IS NULL fondamentalmente indica le posizioni 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 Citta, DataCampione, Precipitazione
  from COMFORT
 where Precipitazione IS NOT NULL;
```

| CITTA | DATAACAMPPI | PRECIPITAZIONE |
|---------------|-------------|----------------|
| SAN FRANCISCO | 21-MAR-93 | .5 |
| SAN FRANCISCO | 22-JUN-93 | .1 |
| SAN FRANCISCO | 23-SEP-93 | .1 |
| SAN FRANCISCO | 22-DEC-93 | 2.3 |
| KEENE | 21-MAR-93 | 4.4 |
| KEENE | 22-JUN-93 | 1.3 |
| KEENE | 22-DEC-93 | 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 NULL 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 paragrafo “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 |
| 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 |

Queste verifiche logiche possono anche essere combinate, 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, e 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; vengono illustrati nel Capitolo 7. È anche possibile consultare il Capitolo 37 per trovare ulteriori informazioni al riguardo.

Test logici su un elenco di valori

Con numeri:

| | |
|-----------------------------|--|
| Pagina IN (1,2,3) | Pagina è nell'elenco (1,2,3) |
| Pagina NOT IN (1,2,3) | Pagina non è nell'elenco (1,2,3) |
| Pagina BETWEEN 6 AND 10 | Pagina è uguale a 6, 10 o qualunque valore tra essi compreso |
| Pagina NOT BETWEEN 6 AND 10 | Pagina è minore di 6 o maggiore di 10 |

Con lettere (o caratteri):

| | |
|---------------------------------|--|
| Sezione IN ('A','C','F') | Sezione è nell'elenco ('A','C','F') |
| Sezione NOT IN ('A','C','F') | Sezione non è nell'elenco ('A','C','F') |
| Sezione BETWEEN 'B' AND 'D' | Sezione è uguale a 'B', 'D' o qualsiasi lettera tra esse compresa (in ordine alfabetico) |
| Sezione NOT BETWEEN 'B' AND 'D' | Sezione è precedente a 'B' o successiva a 'D' (in ordine alfabetico) |

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 |

Il risultato ottenuto non è però quello desiderato. In qualche modo, la pagina 1 della sezione A è stata inclusa nelle righe visualizzate. Apparentemente, la stringa “and Pagina > 2” è stata considerata soltanto per le righe della sezione B. Se si sposta la stringa “and Pagina > 2” in alto verso la metà della frase con where, il risultato cambia, ma è ancora sbagliato:

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

| ARGOMENTO | S | PAGINA |
|--------------|---|--------|
| Editoriali | A | 12 |
| Televisione | B | 7 |
| Vita moderna | B | 1 |
| Film | B | 4 |
| Bridge | B | 2 |

Che cosa accade se si pone la stringa “Pagina > 2” all’inizio? Ancora un risultato sbagliato:

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

| ARGOMENTO | S | PAGINA |
|--------------|---|--------|
| Editoriali | A | 12 |
| Televisione | B | 7 |
| Vita moderna | B | 1 |
| Film | B | 4 |
| Bridge | B | 2 |

Come si spiegano questi risultati? Esiste un modo per ottenere da ORACLE una risposta corretta? Anche se entrambi gli operatori AND e OR sono 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). Ciò significa che la seguente query con where:

```
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:

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

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

3.6 Un altro impiego delle sottoquery con where

Che cosa accadrebbe se gli operatori logici citati nei paragrafi “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.

Ad esempio, si supponga che l'autore dell'articolo di argomento “Salute” pubbli chi i suoi articoli in diversi giornali, ciascuno dei quali gli invia una copia del sommario in cui è incluso il suo pezzo. Naturalmente, l'argomento non viene considerato da tutti i giornali della stessa importanza e 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 collocazione nel giornale locale? Si potrebbe tentare con:

```
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 sottoquery

In effetti, il comando select tra parentesi (definito *sottoquery*) 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 (si faccia riferimento alla Figura 3.2, riportata in precedenza) e non funziona con elenchi di valori, quindi, se la sottoquery 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 sottoquery, a patto che il risultato di queste sia una riga singola. Ad 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 sottoquery

Esattamente come in una sottoquery possono essere utilizzati gli operatori logici a valore singolo, possono essere utilizzati gli operatori a valori multipli. Se una sottoquery ha come risultato una o più righe, il valore nella colonna per ciascuna riga viene visualizzato in un elenco. Ad 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 di localizzazione per tutte le città e i paesi corrispondenti:

```
select Citta, Paese from LOCAZIONE;
```

| 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 LOCAZIONE:

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

| CITTA | PAESE |
|--------|---------|
| PARIGI | FRANCIA |
| SPARTA | GRECIA |

Lo stesso risultato si ottiene impostando una sottoquery, 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 LOCAZIONE
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 sottoquery che produca un elenco che possa essere verificato logicamente. È importante considerare i seguenti aspetti.

- La sottoquery deve avere una sola colonna, oppure deve confrontare le colonne selezionate con le colonne multiple poste tra parentesi nella query principale (per ulteriori informazioni, si consulti il Capitolo 11).
- La sottoquery deve essere racchiusa tra parentesi.
- Le sottoquery che hanno come risultato soltanto una riga possono essere utilizzate sia con operatori a valori singoli, sia con operatori a valori multipli.
- Le sottoquery che hanno come risultato più di una riga possono essere utilizzate soltanto con operatori a valori multipli.
- L'operatore `BETWEEN` non può essere utilizzato con una sottoquery; ovvero, la query seguente:

```
select * from METEO
where Temperatura BETWEEN 16
      AND (select Temperatura
            from METEO
            where Citta = 'PARIGI');
```

non funziona. Tutti gli altri operatori a valori multipli, viceversa, funzionano con le sottoquery.

3.7 Come combinare le tabelle

Tutto quanto si è detto finora vale se tutte le colonne che si vogliono visualizzare si trovano in un'unica tabella, oppure se è possibile effettuare una semplice verifica su un'altra tabella impostando una sottoquery. Tuttavia, 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 LOCAZIONE:

```
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, mancano però informazioni sul tempo. Eppure le tabelle CLIMA e LOCAZIONE hanno una colonna in comune: Citta. È quindi possibile collegare le informazioni delle due tabelle unendole insieme. Occorre semplicemente 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 | O |
| LIMA | PIOGGIA | 7 | 12.03 | S | 77.03 | O |
| MANCHESTER | NEBBIA | 19 | 53.3 | N | 2.15 | O |
| 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 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 LOCAZIONE 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.ColonnaNome”, 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 a 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 “LOCAZIONE.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 query con 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 locazione punto città”, ovvero, in cui la colonna Citta della tabella CLIMA equivale alla colonna Citta nella tabella LOCAZIONE.

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 imponendo 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 LOCAZIONE. Inoltre, delle 14 città della tabella LOCAZIONE, sono presenti in questa tabella soltanto quelle della tabella CLIMA. Utilizzando la query con 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, LOCAZIONE
 where CLIMA.Citta = LOCAZIONE.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 che si possono utilizzare su una tabella comune non possono invece essere utilizzate su una vista, ma sono poche e per lo più riguardano la modifica di righe e l'impostazione di indici di tabelle, argomenti che vengo-

no discussi nei capitoli successivi. Nella maggior parte dei casi, una vista può dare risultati ed essere manipolata esattamente come qualsiasi altra tabella.

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-0704: invalid column name
```

Come mai? 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 i paesi della Grecia.

```
create or replace view INVASIONE as
select METEO.Citta, Condizione, Temperatura, Latitudine,
       NordSud, Longitudine, EstOvest
  from METEO, LOCAZIONE
 where METEO.Citta = LOCAZIONE.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 18.

La relazione logica della query con `where` ora è stata espansa per includere entrambe le tabelle di origine e un test a valore singolo su una colonna in una delle tabelle origine. Ora si può 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 della visualizzazione 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, qualsiasi qualificazione inserita in una query può diventare parte di una vista. Si potrebbe ad 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 istantanee 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.

Ad esempio, si può creare una vista che limiti i valori sulla base dei valori di colonna. Come viene mostrato di seguito, una query che limita la tabella LOCAZIONE alla colonna Paese potrebbe essere utilizzata per limitare le righe visibili nella vista:

```
create or replace view LOCAZIONI_PERU as
select * from LOCAZIONE
where Paese = 'PERU';
```

Se l'utente imposta la query per LOCAZIONI_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 consente la visualizzazione di un valore soltanto se selezionata, ma non è una vera colonna di una tabella. Selezionando la pseudocolonna User, viene sempre visualizzato il nome dell'utente di ORACLE che ha impostato la query. Quindi, se una colonna in una tabella contiene nomi utente, questi valori possono essere verificati con la pseudocolonna User per limitarne le righe, come mostrato nell'esempio seguente, dove viene impostata una query relativa alla 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 viene trattato in maniera più approfondita nel Capitolo 17.

La query con where può essere utilizzata per unire insieme due tabelle che hanno una colonna comune. L'insieme di dati risultante può essere trasposto 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

In ORACLE8 è 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 disponibili in ORACLE8, mostrando l'impatto che hanno sul linguaggio SQL. Le caratteristiche orientate agli oggetti di tipo più avanzato vengono descritte in maniera dettagliata nei capitoli successivi; nel presente capitolo vengono introdotti i concetti e una panoramica generale.

4.1 È obbligatorio utilizzare gli oggetti?

Il passaggio a ORACLE8 non obbliga certo a impiegare concetti orientati agli oggetti (OO, object oriented) 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
 - Relazionale a oggetti
 - Orientato agli oggetti

Il database tradizionale di ORACLE (RDBMS).
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.
Database orientato agli oggetti il cui sviluppo è basato esclusivamente su analisi e progettazione orientate agli oggetti.

Con ORACLE è possibile utilizzare al meglio tutte e tre le diverse implementazioni: relazionale, relazionale a oggetti e orientata agli oggetti. Se si ha già familiarità con ORACLE come database relazionale (l'unica opzione possibile con la versione ORACLE7 e le precedenti), si può continuare a utilizzare ORACLE8 nella stessa maniera. Poiché le funzionalità orientate agli 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 orientate agli oggetti, è possibile scegliere anche questa opzione. Indipendentemente dal metodo prescelto, occorre innanzitutto acquisire familiarità con le funzioni e le caratteristiche specifiche di ORACLE come database relazionale. Anche se si è pianificato di utilizzare soltanto le possibilità di tipo orientato agli 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).

Questa parte del libro è iniziata con una panoramica degli elementi fondamentali del discorso in SQL. In questo capitolo vengono illustrate quelle stesse parti del discorso con l'aggiunta delle strutture relazionali a oggetti. Nei capitoli seguenti le funzioni di ORACLE vengono descritte in maniera più dettagliata, con paragrafi su PL/SQL, trigger e procedure. Scorrendo i capitoli su PL/SQL e seguendo le procedure si troveranno diversi argomenti correlati alle caratteristiche orientate agli oggetti disponibili con ORACLE8. È importante comprendere funzioni, strutture e linguaggio di programmazione di ORACLE prima di implementare le strutture più avanzate di tipo orientato agli oggetti e relazionale a oggetti.

4.2 Perché utilizzare gli oggetti?

Poiché non è indispensabile utilizzare gli oggetti, perché mai lo si dovrebbe fare? Inizialmente può sembrare che l'utilizzo di caratteristiche orientate agli 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. Gli entusiasti di queste caratteristiche orientate agli oggetti sostengono che gli oggetti riducono la complessità consentendo all'utente un approccio intuitivo nella rappresentazione di dati complessi e delle relazioni fra di essi. Ad esempio, se si desidera spostare un'automobile (un oggetto), la si può spostare 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.

- *Riuso degli oggetti.* Se si scrive codice orientato agli 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. Ad esempio, se si creano tipi di dati standard da utilizzare per tutti gli indirizzi, allora 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 possono influire su di esso, quindi si possono unire i dati e i metodi per accedervi. Definendo i percorsi di accesso in questo modo, si possono 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 implementare le caratteristiche. Tuttavia, come si osserverà in questo capitolo, i principi fondamentali per modificare l'RDBMS ORACLE in modo da includere le funzionalità orientate agli oggetti si basano semplicemente sul modello relazionale presentato nei primi capitoli. Il breve tempo impiegato per sviluppare e utilizzare tipi di dati astratti, come viene illustrato successivamente in questo 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. Poiché un oggetto è costituito dalla combinazione di dati e metodi, tutti possiedono degli oggetti. Si consideri nuovamente l'elenco dei lavoratori di Talbot, visualizzato nella Figura 4.1. Talbot ha impostato uno standard per la struttura degli indirizzi: prima il nome della persona, poi il nome dell'abitazione, il nome della strada e il nome della città. Quando viene aggiunta una nuova persona alla lista per Dora, viene seguita la stessa procedura. I metodi utilizzati per questo tipo di dati relativi agli indirizzi potrebbero comprendere i seguenti.

- Aggiungi_Persona Per aggiungere una persona all'elenco.
- Aggiorna_Persone Per aggiornare i dati relativi a una persona.
- Elimina_Persona Per cancellare una persona dall'elenco.
- Conta_alloggio Per contare il numero di lavoratori per alloggio.

Come mostra il metodo Conta_alloggio, 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. Ad esempio, se Talbot avesse memorizzato le date di nascita dei suoi lavoratori, si potrebbe utilizzare un metodo Eta per calcolare e creare report su tali dati. ORACLE supporta molti tipi diversi di oggetti, descritti nei paragrafi seguenti.

| <i>Addresses for Dora's Help</i> | |
|---|---|
| Bart Sargent, Cannons Retreat House, Well St, Berkley | |
| Pet Lavery, Rose Hill Rd, N. Edmenton | 1 |
| Dick Jones, Cypress Hotel, N. Edmenton | |
| * Arch Talbot, Papa King Rooming, 127 Main, N. Edmenton | |
| Freddie Lee, Rose Hill Rd, N. Edmenton | |
| Hay and Hume Wallbom, Tool Lake Rose Hill | |
| * Elbert Talbot, Weilbricht Rooming, 320 Geneva, Neene | |
| Richard Weiland, brother, Weilbricht Rooming | 2 |
| ? Peter Lawson, Cannons Retreat House, Hill St, Berkley | |
| Joe Hopkins, Matt's Long Barn House, 3 Mile Rd, Neene | |
| Helen Brandt, Well Station, N. Edmenton | |
| William Seving, Cannons Retreat House, Well St, Berkley | |
| George Oscar, Rose Hill, 46 Pet Lavery | |
| Donald Rolfe, Matt's Long Barn House, 3 Mile Rd, Neene | |
| Geehardt Kenton, Papa King Rooming, 127 Main, Edmenton | |
| Clyde Hannan, Bell's Dairy Farm, Neene | |
| Wilfred Lovell ? | |
| Roland Harrel, Matt's Long Barn House, 3 Mile Road, Neene | |
| Danielle Lawson (with Peter at Cannons) | |
| George S. Macwick and Lelly (Lily?) with Wallbom | |
| Alice Butler, 46 Goldenrod Lane, Neene | |
| Dick Jones, Co. | |
| Andrew Schuster, Rd 1, Ryedale | |

Figura 4.1 Il registro degli indirizzi di G.B. Talbot.

Tipi di dati astratti

I *tipi di dati astratti* sono tipi di dati costituiti da uno più sottotipi. Non avendo i vincoli dei tipi di dati standard per ORACLE, NUMBER, DATE e VARCHAR2, con i tipi di dati astratti è possibile descrivere i dati in maniera più accurata.

Ad esempio, un tipo di dato astratto per gli indirizzi può essere formato 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.

Due dei vantaggi elencati precedentemente relativamente all'utilizzo di oggetti, il riuso 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 posizioni multiple, si può essere certi che gli stessi dati logici vengano rappresentati nella stessa maniera in quelle posizioni. Il riuso di un tipo di dati astratto rafforza la rappresentazione standardizzata dei dati.

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 una tabella all'interno di un'altra tabella. Una tabella di questo tipo 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 righe multiple.

In un certo senso si tratta di un modo per memorizzare relazioni multiple all'interno di una tabella. Si consideri ad esempio una tabella che contenga informazioni sui settori di una azienda, in cui ciascun settore 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 dei progetti 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 aggregazioni sono riportati nel Capitolo 26.

Array variabili

Un *array variabile* è, come una tabella annidata, un'aggregazione; si tratta di un insieme di oggetti dello stesso tipo di dati. 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 insieme di righe predefinito.

Gli array variabili, detti anche VARRAY, consentono di memorizzare nelle tabelle attributi ripetuti. Ad 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 progetti multipli. In una 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.

Si possono invece 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. Esempi di array variabili e altri tipi di aggregazioni sono riportati nel Capitolo 26.

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 si possono memorizzare dati formati da caratteri fino a una lunghezza di 4 GB. Il tipo di dati NCLOB viene utilizzato per memorizzare dati CLOB per gruppi di caratteri multipli. 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 si trovano nel sistema operativo; il database contiene soltanto un indicatore per questi file. Le dimensioni dei file esterni sono limitate soltanto dal sistema operativo. Poiché i dati sono conservati all'esterno del database, non è possibile mantenerne la corrispondenza o l'integrità.

Si possono impostare dati LOB multipli per ciascuna tabella. Ad 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. Per una descrizione dettagliata delle modalità di creazione e di utilizzo di questi tipi di dati, si può consultare il Capitolo 27.

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 essenzialmente indicatori di oggetti riga. Un “oggetto riga” è diverso da un “oggetto colonna”. Un esempio di “oggetto colonna” potrebbe essere un array variabile: si tratta di 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 nel Capitolo 31. Come si è osservato precedentemente in questo capitolo, non è necessario utilizzare tutte le funzioni orientate agli oggetti disponibili in ORACLE. I riferimenti sono solitamente tra le ultime caratteristiche orientate agli oggetti installate quando si passa da un’applicazione relazionale a un’applicazione relazionale a oggetti o a un approccio orientato agli oggetti.

Viste oggetti

Le *viste oggetti* consentono di aggiungere concetti orientati agli oggetti a tabelle relazionali esistenti. Ad esempio, è possibile creare un tipo di dati astratto sulla base di una definizione di tabella già esistente. Quindi, 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 orientate agli 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 al Capitolo 25.

Caratteristiche degli oggetti

Un oggetto ha un nome, una rappresentazione standard e un insieme standard di operazioni che agiscono su di esso, dette *metodi*. Perciò un tipo di dati astratto ha un nome, una rappresentazione standard e dei metodi definiti per accedere ai dati. Tutti gli oggetti che utilizzano il tipo di dati astratto condividono la stessa struttura, gli stessi metodi e la stessa rappresentazione. I tipi di dati astratti modellano *classi* di dati all’interno del database.

I tipi di dati astratti fanno parte di un concetto orientato agli oggetti chiamato *astrazione*, che indica l’esistenza concettuale di classi all’interno del database. Come si è detto in precedenza in questo capitolo, i tipi di dati astratti possono essere annidati. Non è necessario creare tabelle fisiche a ogni livello di astrazione, perché basta l’esistenza strutturale dei tipi astratti. I metodi collegati a ogni livello di astrazione possono essere richiamati dai livelli più elevati, perciò è possibile accedere ai metodi del tipo di dati INDIRIZZO in una chiamata al tipo di dati PERSONA.

Alla creazione, gli oggetti ereditano le strutture degli elementi di dati da cui discendono. Si consideri l'esempio di un'automobile citato in precedenza. Se "automobile" è una classe di dati, allora "fuoristrada", un tipo di automobile, eredita le definizioni strutturali di "automobile". Dalla prospettiva dei dati, i tipi di dati astratti ereditano le rappresentazioni dei loro "genitori". La possibilità di creare gerarchie di tipi di dati astratti è disponibili con ORACLE8.1.

Oltre a ereditare strutture di dati, le classi possono ereditare il comportamento delle loro classi "genitore"; questo concetto si chiama *ereditarietà di implementazione*. In pratica è possibile eseguire un metodo su un oggetto anche se quest'ultimo non lo utilizza, purché il metodo in questione sia definito per una delle classi "genitore" dell'oggetto corrente. Se è possibile accedere a una struttura dati soltanto tramite una serie di metodi definiti, questa struttura si dice *incapsulata* dai metodi. In un sistema encapsulato, i metodi definiscono tutti i percorsi di accesso ai dati. Tuttavia, l'implementazione orientata agli oggetti di ORACLE si base su un sistema relazionale, in cui i mezzi di accesso ai dati non sono mai del tutto limitati, perciò i dati in ORACLE non possono mai essere del tutto encapsulati. È possibile approssimare l'incapsulamento limitando l'accesso alle tabelle e facendo in modo che tutti gli accessi siano realizzati tramite procedure e funzioni, ma questo impedisce di sfruttare la piena potenza di un database relazionale.

Si considerino le query su LOCAZIONE e CLIMA eseguite nel Capitolo 1. Se i dati CLIMA fossero stati pienamente encapsulati, non si sarebbe potuto scrivere la semplice query che riportava i dati climatici di diverse città. Come si è detto in precedenza in questo capitolo, ORACLE supporta i modelli relazionale, relazionale a oggetti e orientato agli oggetti, ma poiché supporta tutti questi modelli attraverso un motore relazionale, e poiché l'incapsulamento viola i concetti fondamentali dei database relazionali, questo è il concetto orientato agli oggetti più difficile da far valere in ORACLE.

Attualmente ORACLE8 non supporta un concetto orientato agli oggetti detto *polimorfismo*, che consiste nella possibilità, per diversi oggetti, di interpretare diversamente le stesse istruzioni. Ad esempio, si supponga che l'istruzione sia "accendere"; questa istruzione, applicata all'oggetto "sigaretta" sarebbe interpretata in un modo, mentre applicata all'oggetto "dibattito" sarebbe intrerpretata in un modo totalmente diverso.

• •

Convenzioni di denominazione per gli oggetti

Al termine del Capitolo 2 è stato brevemente 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. Nel Capitolo 35 vengono illustrate in maniera più completa le regole per le convenzioni di denominazione, comunque negli esempi di procedure orientate agli oggetti citati in questo libro vengono applicate le regole seguenti.

1. I nomi di tabella e di colonna devono essere singolari (come nel caso di IMPIEGATO, Nome e Prov).

2. I tipi di dati astratti devono essere sostantivi singolari con suffisso in _TY (come per PERSONA_TY o INDIRIZZO_TY).
3. I nomi di tabella e di tipo di dati devono sempre essere in caratteri maiuscoli (come IMPIEGATO o PERSONA_TY).
4. I nomi di colonna devono sempre iniziare con lettera maiuscola (come Prov e _Data).
5. Le viste oggetti devono essere sostantivi singolari con suffisso in _OV (come PERSONA_OV o INDIRIZZO_OV).
6. I nomi di tavole annidate sono sostantivi plurali con suffisso in _NT (come LAVORATORI_NT).
7. I nomi di array variabili devono essere sostantivi plurali con suffisso in _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 comune che si trova nella gran parte dei sistemi: l'indirizzo. Anche in un sistema semplice come quello di Talbot gli indirizzi vengono conservati e selezionati. Nel caso del libro mastro originario di Talbot, venivano mantenuti gli indirizzi dei lavoratori (Figura 4.1), strutturati secondo un formato standard: nome dell'abitazione, seguito dalla via e dalla città.

Questo schema 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 la provincia (o lo stato, nel caso di indirizzi degli Stati Uniti) e 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),
   Citta  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, Citta, 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 il Capitolo 25 e la voce create type body nel Capitolo 37.

Si osservi l'espressione `as object` all'interno del comando `create type`. Con questa dicitura viene identificato esplicitamente `INDIRIZZO_TY` come un'implementazione di tipo orientato agli oggetti.

Ora che è stato creato il tipo di dati `INDIRIZZO_TY`, è possibile utilizzarlo nell'ambito di altri tipi di dati. Ad esempio, Talbot avrebbe potuto decidere di creare un tipo di dati standard per le persone. Le persone hanno nomi e indirizzi, perciò Talbot avrebbe potuto creare il tipo di dati 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. La stringa:

```
(Nome      VARCHAR2(25),
```

definisce la prima colonna della rappresentazione di `PERSONA_TY`. La stringa successiva:

```
INDIRIZZO  INDIRIZZO_TY);
```

definisce la seconda colonna. Per la seconda colonna, `INDIRIZZO`, viene utilizzato il tipo di dati astratto `INDIRIZZO_TY` creato precedentemente. 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);
```

Pertanto, una voce di `PERSONA_TY` è composta dalle colonne Nome, Via, Città, 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. Ad 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 questi 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).

Talbot ora avrebbe potuto 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 qualsiasi tipo di dati creato.

Per memorizzare dei dati, occorre creare una tabella che utilizzi il tipo di dati desiderato; solo allora sarà possibile memorizzare dei dati in quella tabella, formattata per il tipo di dati definito.

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 | | NAMED TYPE |

La colonna Persona viene illustrata attraverso il comando describe come tipo di dati “named type”. Con questo comando non viene visualizzato il nome del tipo di dati associato con la colonna Persona. Per ottenere questa informazione, occorre impostare una query direttamente al dizionario di dati.

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 il Capitolo 32 per una guida “orientata all’utente” riguardo alle viste del dizionario dati disponibili.*

Si possono impostare delle query per il dizionario di dati USER_TAB_COLUMNS al fine di visualizzare i tipi di dati associati a ogni colonna nella tabella CLIENTE:

```
select Column_Name,
       Data_Type
  from USER_TAB_COLUMNS
 where Table_Name = 'CLIENTE';
```

| COLUMN_NAME | DATA_TYPE |
|-------------|-----------|
| CLIENTE_ID | NUMBER |
| PERSONA | PERSON_TY |

NOTA *In USER_TAB_COLUMNS sono dettagliate le informazioni relative alle colonne e alle viste di proprietà dell’utente.*

Nel risultato della query viene evidenziato 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.

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 | | NUMBER |

Non accade spesso di dover completamente scomporre i tipi di dati che costituiscono una tabella; tuttavia, se è necessario farlo, si possono utilizzare 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, si possono 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 proprietà dell'utente o a cui l'utente può accedere. Sia ALL_TAB_COLUMNS sia ALL_TYPE_ATTRS contengono una colonna denominata Owner in cui sono identificati i proprietari delle tabelle o dei tipi di dati. Per la descrizione completa delle viste del dizionario dati, si rimanda al Capitolo 32.*

Inserimento di record nella tabella CLIENTE

Con ORACLE, quando viene creato un tipo di dati astratto, vengono creati dei metodi detti *costruttori* per la gestione dei dati. 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 possono impiegare tali metodi. Ad esempio, nella tabella CLIENTE è utilizzato il tipo di dati PERSONA_TY e per il tipo di dati PERSONA_TY viene utilizzato il tipo di dati INDIRIZZO_TY. Per inserire un record nella tabella CLIENTE è necessario utilizzare i tipi di dati PERSONA_TY e INDIRIZZO_TY. Per inserire dei record utilizzando questi tipi di dati occorre utilizzare i metodi costruttori appropriati.

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 alle 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 metodo `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 esattamente nello stesso ordine degli attributi del tipo di dati.

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

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

Ora è stato inserito il secondo record nella tabella CLIENTE. Basta utilizzare i metodi 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, occorre semplicemente 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 normale. Tuttavia, quando si impostano query in relazione a tutte le colonne della tabella CLIENTE, si rivela la complessità dei tipi di dati utilizzati:

```
select *  
  from CLIENTE;
```

```
CLIENTE_ID
```

```
-----
```

```
PERSONA(NOME, INDIRIZZO(VIA, CITTA, PROV, CAP))
```

```
-----  
 1
```

```
PERSONA_TY('NEIL MULLANE', INDIRIZZO_TY('57 MT PLEASANT ST', 'FINN',  
                                         'NH', 11111))
```

```
 2
```

```
PERSONA_TY('SEYMOUR HESTER', INDIRIZZO_TY('1 STEPAHEAD RD', 'BRIANT',  
                                         'NH', 11111))
```

Nel risultato della query viene indicato che Cliente_ID è una colonna all'interno della tabella CLIENTE e che per la colonna Persona è stato impostato un tipo di dati astratto.

Il nome di colonna Persona indica i nomi dei tipi di dati utilizzati e l'annidamento del tipo di dati INDIRIZZO_TY all'interno del tipo di dati PERSONA_TY.

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
```

Nome infatti non è una colonna all'interno della tabella CLIENTE, ma un attributo nell'ambito del tipo di dati astratto PERSONA_TY. Per visualizzare i valori di Nome, occorre perciò impostare una query per l'attributo Nome all'interno della colonna Persona. Non è possibile interrogare direttamente PERSONA_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, Persona.Nome
  from CLIENTE;
```

Si osservi la sintassi per la colonna Nome:

Persona.Nome

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:

Colonna.Attributo

Ciò può creare un po' di confusione. Nel comando insert è stato utilizzato il nome del tipo di dati (in realtà è stato utilizzato il nome del metodo costruttore, che equivale al nome del tipo di dati). Nel comando select è stato invece utilizzato il nome della colonna.

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 parametro Colonna.Attributo in modo da comprendere il tipo di dati annidato. Il formato corretto è il seguente:

Colonna.Colonna.Attributo

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

```
select Persona.INDIRIZZO.Via
  from CLIENTE;
```

PERSONA.INDIRIZZO.VIA

57 MT PLEASANT ST
1 STEPAHEAD RD

La sintassi:

```
select Persona.INDIRIZZO.Via
```

indica a ORACLE esattamente dove trovare l'attributo Via.

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. Ad esempio, non è possibile selezionare i valori per Citta della tabella CLIENTE senza sapere che Citta fa parte dell'attributo INDIRIZZO e INDIRIZZO fa parte della colonna Persona. Non è possibile inserire o aggiornare (con il 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:

```
select Persona.Nome,  
       Persona.INDIRIZZO.Citta  
  from CLIENTE  
 where 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 come mostrato negli esempi precedenti. Ad 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  
   set Persona.INDIRIZZO.Citta = 'HART'  
 where Persona.INDIRIZZO.Citta = 'BRIANT';
```

La clausola where viene utilizzata da ORACLE per trovare i record corretti da aggiornare e quella con set 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. Nel Capitolo 25 vengono presentati ulteriori impieghi dei tipi di dati astratti; tabelle annidate e array variabili sono invece descritti più dettagliatamente nel Capitolo 26.

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. Ad 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.

Ad 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 tabelle multiple. Se il tipo di dati astratto viene utilizzato per tabelle relazionali multiple, si può ottenere il vantaggio del riuso degli oggetti e dell'aderenza alla definizione standard delle strutture di attributi.

Una volta definiti i propri 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 più probabilmente 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 il Capitolo 25.

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. Nel Capitolo 25 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, od 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.

Nei capitoli seguenti vengono descritte in maniera dettagliata le implementazioni in ORACLE del linguaggio SQL. Questi capitoli sono improntati sull'implementazione di SQL per tabelle relazionali; le stesse funzioni non sono adeguate per tabelle con caratteristiche orientate agli oggetti come i tipi di dati astratti. Scorrendo i capitoli sull'utilizzo di SQL, si trovano capitoli specifici su PL/SQL, l'estensione procedurale di ORACLE a SQL. 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 che seguono i capitoli su PL/SQL.

Seguendo le spiegazioni dei capitoli sulle modalità relazionali a oggetti (in cui sono illustrati metodi, viste oggetti, tabelle annidate, array variabili e LOB), si trovano capitoli specifici su altre applicazioni aggiuntive di ORACLE. Alcune di queste caratteristiche comprendono la replicazione di dati, lo strumento ConText (per la ricerca di testi), e caratteristiche orientate agli oggetti avanzate. 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. Nel prossimo capitolo viene presentata un'introduzione a SQLPLUS, lo strumento di ORACLE per operare in maniera interattiva con SQL.

• Capitolo 5

• Report e comandi fondamentali di SQLPLUS

- 5.1 Creazione di un report semplice
- 5.2 Altre caratteristiche
- 5.3 Controllo dell'ambiente SQLPLUS
- 5.4 I fondamenti

SQLPLUS viene generalmente considerato come una sorta di compilatore di report interattivo. Con questo strumento viene utilizzato SQL per ottenere informazioni dal database di ORACLE ed è possibile creare report eleganti e ben formattati perché si possono gestire facilmente titoli, intestazioni di colonna, subtotali e totali, riformattazione di numeri e testo, e molto altro ancora. Può anche essere utilizzato per modificare il database utilizzando 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. Questa proprietà poco conosciuta viene trattata in dettaglio nel Capitolo 20.

SQLPLUS viene utilizzato molto più comunemente per semplici query e report stampati. Ottenere con SQLPLUS la formattazione delle informazioni in un report secondo il proprio gusto e le proprie esigenze richiede soltanto qualche comando o parola chiave che dia istruzioni a SQLPLUS su come operare. Questi comandi sono elencati nella Tabella 5.1 e descritti dettagliatamente nel Capitolo 37.

In questo capitolo viene presentato un report di base compilato utilizzando SQLPLUS e vengono 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 sperimentate le fasi della procedura, queste risulteranno semplici da comprendere e diventeranno presto familiari.

Tabella 5.1 Comandi di base di SQLPLUS. (*continua*)

| 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. |

Tabella 5.1 Comandi di base di SQLPLUS.

| COMANDO | DEFINIZIONE |
|----------------|---|
| 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 su un file un report che normalmente si vedrebbe visualizzato sullo schermo in un file, in modo che si possa stamparlo. |
| /**/ | Contrassegna l'inizio e la fine di un commento in una query SQL. 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. 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 | 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. |

Si possono compilare report con SQLPLUS mentre si lavora in maniera interattiva, ovvero si possono digitare comandi riferiti a intestazioni di pagina, titoli di colonna, formato, interruzioni, totali e così via, quindi eseguire una query SQL, e con SQLPLUS viene immediatamente prodotto il report formattato secondo le indicazioni specificate. Si tratta di un approccio interessante nel caso di risposte veloci a domande semplici che non hanno probabilità di essere riproposte. Tuttavia, capita più spesso di voler ottenere comuni report complessi che devono essere prodotti periodicamente, e che è necessario stampare oltre che visualizzare semplicemente sullo schermo. Purtroppo, quando si esce da SQLPLUS vengono immediatamente dimenticate tutte le istruzioni impostate. Se si utilizza SQLPLUS soltanto in questa maniera interattiva, eseguire lo stesso report successivamente significa inserire nuovamente tutti i dati.

L'alternativa consiste semplicemente nel digitare i comandi, riga dopo riga, in un file. Successivamente è possibile leggere questo file con SQLPLUS come se fosse uno script ed eseguire i comandi proprio come se venissero digitati al momento. In effetti, così facendo viene creato un programma di generazione di report senza l'aiuto di programmatore o compilatori.

Il file è creato utilizzando uno qualsiasi dei comuni editor disponibili, o anche (con alcune limitazioni) con un elaboratore di testi.

L'editor non fa parte di ORACLE. Questi programmi sono disponibili in centinaia di varietà e ognuno ha il suo preferito. ORACLE ha compreso questa situazione e ha deciso di consentire all'utente di scegliere l'editor da utilizzare, invece di fornire un editor integrato e obbligare gli utenti a utilizzarlo. Quando l'utente è pronto per utilizzare il suo editor, deve 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 di cui lo si era lasciato ed eseguire quel report. Si osservi in proposito la Figura 5.1.

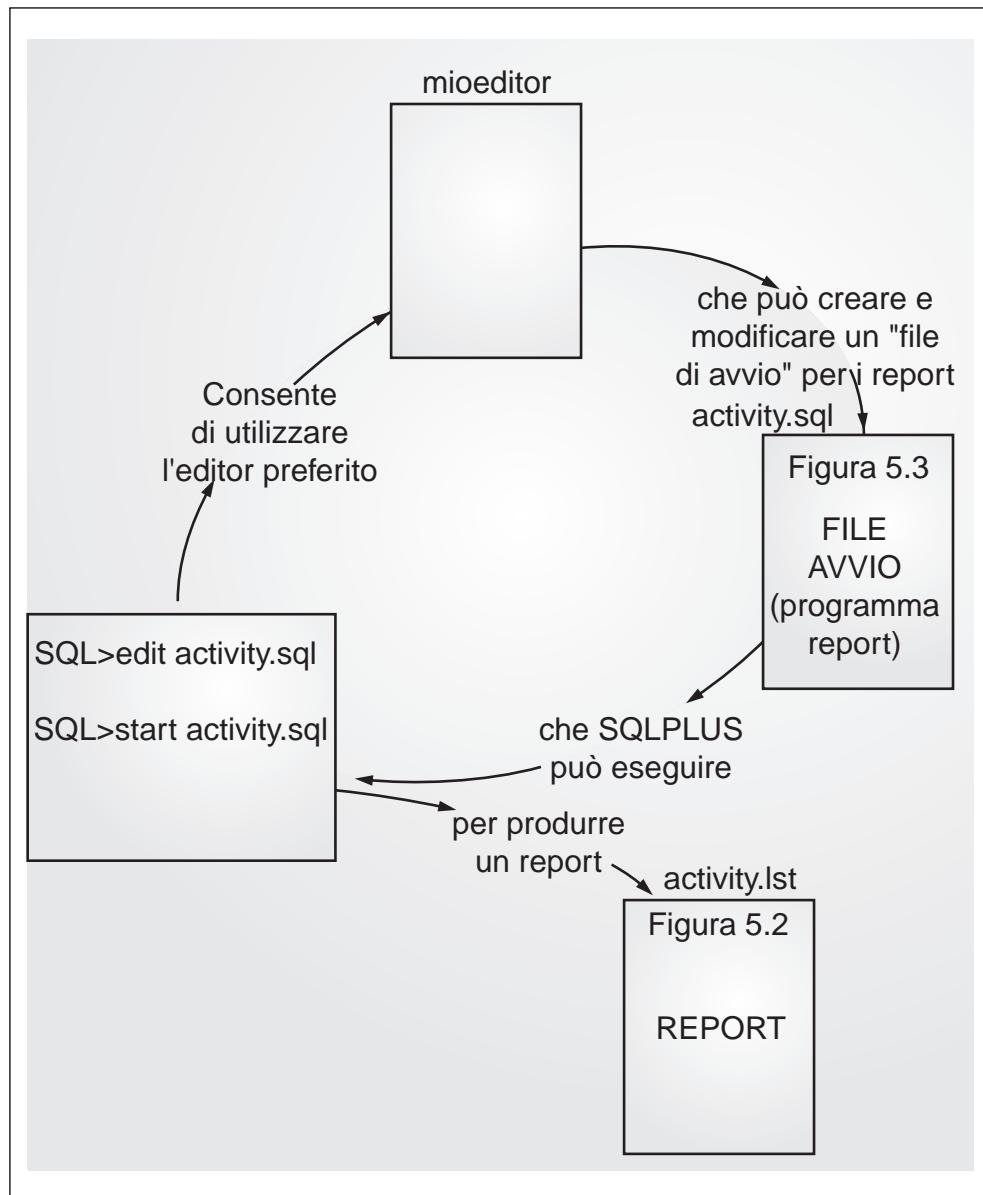


Figura 5.1 SQLPLUS consente di utilizzare l'editor preferito per creare programmi di report.

In SQLPLUS è disponibile anche un piccolo editor preinstallato, talvolta definito *editor della riga di comando*, che consente di modificare velocemente una query SQL senza dover lasciare SQLPLUS. L'utilizzo di questa caratteristica viene illustrato più avanti in questo capitolo.

5.1 Creazione di un report semplice

Nella Figura 5.2 è illustrato un report semplice e veloce prodotto per G.B. Talbot, in cui sono dettagliati gli articoli venduti durante la seconda metà del 1901 e l'entrata relativa a quel periodo.

Nella Figura 5.3 è mostrato 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
```

NOTA *Come distinguere SQLPLUS e SQL: l'istruzione di selezione visibile verso la fine della Figura 5.3, che inizia con il termine "select" e finisce con un punto e virgola (;), è espresso in SQL, la lingua utilizzata per dialogare con il database di ORACLE. Ogni altro comando visibile nella pagina è un comando SQLPLUS, utilizzato per formattare i risultati di una query SQL in un report. In questo modo il file ACTIVITY.SQL viene letto da SQLPLUS e vengono eseguite 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. A seconda dell'esperienza dell'utente, questo file può apparire avanzato o di livello elementare; è costituito da una serie di istruzioni semplici per SQLPLUS.*

remark

Le prime cinque righe della Figura 5.3, contrassegnate dal numero 1, costituiscono la documentazione relativa al file stesso. Queste righe iniziano con le lettere:

```
rem
```

abbreviazione di remark. In SQLPLUS viene ignorata qualsiasi riga che inizia con queste lettere, il che consente di aggiungere commenti, annotazioni e spiegazioni per qualsiasi file di avvio creato.

È sempre opportuno introdurre queste note 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 sullo scopo del file.

Si tratta di un accorgimento che si rivela indispensabile, quando iniziano ad accumularsi decine di report.

| Vendite per prodotto durante il 1901 Secondo semestre (lug-dic) | | | | | | |
|--|------------------------|---------------------|------|------|----------|-------|
| Data | Venduto a | Bene Venduto | Quan | Tipo | TASSO | Pro |
| 15-OCT-01 | GENERAL STORE | MANZO | 935 | LB | 0.03 | 28.05 |
| 15-NOV-01 | FRED FULLER | | 37 | LB | 0.04 | 1.48 |
| 21-NOV-01 | ROLAND BRANDT | | 116 | LB | 0.06 | 6.96 |
| 22-NOV-01 | GERHARDT KENTGEN | | 118 | LB | 0.06 | 7.08 |
| ***** | | | | | | |
| | | sum | | | | 43.57 |
| 03-OCT-01 | GARY KENTGEN | STIVALI PER CAVALLI | 1 | EACH | 12.50 | 12.50 |
| 11-NOV-01 | PAT LAVEY | | 1 | EACH | 6.00 | 6.00 |
| ***** | | | | | | |
| | | sum | | | | 18.50 |
| 29-AUG-01 | GERHARDT KENTGEN | BURRO | 5 | LB | 0.23 | 1.15 |
| 11-NOV-01 | PAT LAVEY | | 1 | LB | 0.150.15 | |
| 16-NOV-01 | VICTORIA LYNN | | 5 | LB | 0.16 | 0.80 |
| 18-NOV-01 | JOHN PEARSON | | 6 | LB | 0.16 | 0.96 |
| ***** | | | | | | |
| | | sum | | | | 3.06 |
| Data | Venduto a | Bene Venduto | Quan | Tipo | TASSO | Pro |
| 25-JUL-01 | SAM DYE | VITELLO | 1 | EACH | 1.00 | 1.00 |
| ***** | | | | | | |
| | | sum | | | | 1.00 |
| 03-JUL-01 | SAM DYE | LATTE | 1 | EACH | 35.00 | 35.00 |
| 12-OCT-01 | GEORGE B. MCCORMICK | | 1 | EACH | 35.00 | 35.00 |
| 10-NOV-01 | PAT LAVEY | | 1 | EACH | 28.00 | 28.00 |
| 14-NOV-01 | MORRIS ARNOLD | | 1 | EACH | 35.00 | 35.00 |
| 20-NOV-01 | PALMER WALBOM | | 1 | EACH | 30.00 | 30.00 |
| ***** | | | | | | |
| | | sum163.00 | | | | |
| dal registro di G. B. Talbot | | | | | | |

Figura 5.2 Report SQLPLUS per G.B. Talbot.

```

rem      Nome: activity.sql    Tipo: file di avvio report
rem  Scritto da: G. Koch
rem
rem Descrizione: Report delle vendite per prodotto di G. B. Talbot
rem                  durante la seconda metà del 1901. 1

set headsep ! 2

ttitle 'Vendite per prodotto durante il 1901!Secondo semestre (lug-dic)'
btitle 'dal registro di G. B. Talbot' 3

column Articolo heading 'Bene!Venduto'
column Articolo format a18
column Articolo truncated 4

column Persona heading 'Venduto a' format a18 word_wrapped 5
column Tasso format 90.99
column DataAzione heading 'Data'
column TipoQuantita heading 'Tipo' format a8 truncated
column TipoQuantita heading 'Quan' format 9990
column Pro format 990.99 6
break on Articolo skip 2 7
compute sum of Pro on Articolo 8

set linesize 79
set pagesize 50 9
set newpage 0 10

spool activity.1st 11

select DataAzione, Persona, Articolo, Quantita, TipoQuantita,
       Tasso, Quantita * Tasso "Pro"
  from REGISTRO
 where Azione = 'VENDUTO'          /* solo negli ultimi 6 mesi */
       and DataAzione BETWEEN TO-DATE('01-JUL-1901','DD-MON-YYYY')
                           and TO-DATE('31-DEC-1901','DD-MON-YYYY')
   order by Articolo, DataAzione; 12

spool off

```

Figura 5.3 File di avvio ACTIVITY.SQL utilizzato per produrre il report.

set headsep

La punteggiatura della riga `set headsep` (“heading separator”, separatore di intestazione) contrassegnata dal numero 2 nella Figura 5.3, indica a SQLPLUS come si intende segnalare il punto in cui si desidera interrompere un titolo di pagina o un’intestazione di colonna più lunghi di una singola riga.

Quando si attiva per la prima volta SQLPLUS, il carattere di default per headsep è la barra verticale (|), ma in alcune tastiere questo carattere non è disponibile. Se la tastiera utilizzata non ha questo carattere, o se si preferisce un carattere diverso (come il punto esclamativo utilizzato in questo esempio), è possibile selezionare qualsiasi carattere presente sulla tastiera.

```
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 maniera inaspettata.*

tttitle e btitle

Nella riga contrassegnata dal numero 3 viene subito visualizzata la modalità di utilizzo di headsep. In questa riga:

```
tttitle 'Vendite per prodotto durante il 1901!Secondo semestre (lug-dic)'
```

vengono date istruzioni a SQLPLUS per impostare questo titolo all'inizio (top title, titolo di testa) di ciascuna pagina del report. Con il punto esclamativo tra “1901” e “Secondi” viene prodotta l'interruzione di riga visualizzata nella Figura 5.2. Il titolo prescelto deve essere racchiuso tra apici singoli. Questa riga:

```
btitle 'dall"antico libro mastro di G.B. Talbot'
```

funziona in maniera analoga alla precedente con tttitle, tranne per il fatto che il titolo deve essere posto in fondo alla pagina (“b” sta per “bottom”, in fondo); anche questa notazione deve essere posta tra apici singoli. Si noti la coppia di apici singoli dopo “dall” e si osservi l'effetto prodotto nella Figura 5.2. Poiché gli apici singoli vengono utilizzati per circoscrivere l'intero titolo, un apostrofo (stesso carattere sulla tastiera) risulta ingannevole per SQLPLUS, essendo interpretato come fine del titolo. Per evitare questa circostanza, si possono digitare di seguito due apici singoli quando si desidera inserire un apostrofo. Poiché sia per SQL sia per SQLPLUS gli apici singoli racchiudono stringhe di caratteri, questa tecnica viene utilizzata per entrambi ogni volta in cui è necessario stampare o visualizzare un apostrofo. Tuttavia, questa prassi può generare confusione anche negli utenti più esperti, quindi è meglio evitare l'uso di vocaboli con l'apostrofo.

Quando si utilizza tttitle nel modo illustrato, il titolo viene sempre centrato in base alle dimensioni di riga (linesize; questo argomento viene trattato più avanti in questo paragrafo) e vengono automaticamente inseriti giorno, mese e anno in cui viene eseguito il report nell'angolo superiore sinistro e il numero di pagina nell'angolo superiore destro.

È anche possibile utilizzare tttitle e btitle in maniera più sofisticata, in modo da controllare autonomamente la posizione del titolo e comprendere all'interno del titolo stesso informazioni variabili dalla propria query. Queste tecniche vengono descritte nel Capitolo 13. Come in ORACLE7.2, anche con ORACLE8 si possono utilizzare i comandi repheader e repfooter per creare intestazioni e più di pagina per i report. Si consulti il Capitolo 37 alle voci REPHEADER e REPFOOTER.

column

Con column si possono modificare l'intestazione e il formato di qualsiasi colonna riportata in una stringa con il comando select. Si osservi il report nella Figura 5.2: la terza colonna si chiama in realtà Articolo nel database e nella prima riga con il comando select in fondo alla Figura 5.3. Si osservi ora il punto 4. Con la riga:

```
column Articolo heading 'Bene!Venduto'
```

alla colonna viene assegnata una nuova intestazione. Questa intestazione, come il titolo del report, è suddivisa su due righe perché al suo interno è stato impostato il carattere di separazione (!). Con la riga:

```
column Articolo format a18
```

viene impostata l'ampiezza di visualizzazione a 18. La "a" in "a18" indica che si tratta di una colonna di caratteri alfabetici e non numerici. L'ampiezza può essere impostata virtualmente con qualsiasi valore, senza considerare con che modalità è stata definita la colonna nel database. Se si chiede la descrizione della tabella REGISTRO, dalla quale deriva questo report, viene visualizzato quanto segue:

```
describe REGISTRO
```

| Name | Null? | Type |
|--------------|-------|--------------|
| DATAAZIONE | | DATE |
| AZIONE | | VARCHAR2(8) |
| ARTICOLO | | VARCHAR2(30) |
| QUANTITA | | NUMBER |
| QUANTITATIPO | | VARCHAR2(10) |
| TASSO | | NUMBER |
| IMPORTO | | NUMBER(9,2) |
| PERSONA | | VARCHAR2(25) |

La colonna Articolo è definita con ampiezza pari a 30 caratteri, quindi è possibile che un articolo abbia più di 18 caratteri. Se non si modifica la definizione della colonna per il report, qualsiasi voce con lunghezza superiore a 18 caratteri (come "STIVALI PER CAVALLI") prosegue nella riga successiva, come in questo caso:

```
Bene  
Venduto  
-----  
STIVALI PER CAVALL  
I
```

Il risultato è piuttosto stravagante. Per rimediare, viene inserita un'altra istruzione per la colonna:

```
column Articolo truncated
```

Con questa indicazione viene eliminato ogni carattere oltre l'ampiezza specificata nella riga "column Articolo format a18". Si osservi, nella Figura 5.2, la riga "Bene Venduto" corrispondente al 3 ottobre 1901: la "I" di "CAVALLI" è stata eliminata.

Anche se le tre istruzioni di colonna fornite al punto 4 per la colonna Articolo sono state collocate su tre righe diverse, è possibile unirle in un'unica riga, come avviene al punto 5:

```
column Persona heading 'Venduto a' format a18 word_wrapped
```

Questa istruzione è molto simile a quella utilizzata per la colonna Articolo, tranne per il fatto che l'espressione "word_wrapped" (a capo) sostituisce "truncated". Si osservi la voce "GEORGE B. MCCORMICK" nella Figura 5.2 (relativa alla data del 12 ottobre); era troppo lunga per stare nei 18 spazi consentiti da questo comando di colonna, quindi è stata fatta proseguire sulla seconda riga, ma la suddivisione è stata effettuata mantenendo le parole intere. Questo è l'effetto del comando word_wrapped.

Al punto 6 della Figura 5.3 viene mostrato un esempio di formattazione di tipo numerico:

```
column Tasso format 90.99
```

In questo modo la colonna è stata definita per contenere quattro spazi e un punto decimale. Se però si contano gli spazi della colonna Tasso nel report, si osserva che sono sei. Se invece si considera soltanto il comando column si è portati a credere che la colonna debba avere ampiezza pari a cinque spazi, ma in questo modo non vi sarebbe spazio per un eventuale segno meno, per questo motivo con i numeri viene sempre previsto uno spazio in più a sinistra. Lo "0" in "90.99" indica che per un numero inferiore a 1, come 0.99, viene inserito uno zero invece dello spazio vuoto in quella posizione: "0.99", come si può notare nel report della Figura 5.2.

Nel punto 7 della Figura 5.3 si fa riferimento a una colonna che non appariva nella tabella quando è stato chiesto a SQLPLUS di descriverla:

```
column Pro format 990.99
```

Che cosa significa Pro? Si osservi il comando select nella parte inferiore della Figura 5.3; Essa appare nella riga:

```
Quantita * Tasso "Pro"
```

in cui si indica a SQL di moltiplicare la colonna Quantita per la colonna Tasso e di considerare il risultato come se fosse una nuova colonna denominata Pro. Nella tenuta dei libri contabili, una quantità moltiplicata per un tasso viene spesso definita "proroga". Ciò che risulta quindi evidente in questo caso è la velocità di calcolo di SQL e la sua capacità di rinominare un calcolo con un nome di colonna più semplice (Pro come abbreviazione di "proroga"). Conseguentemente Pro viene considerata come una nuova colonna e tutte le opzioni di formattazione e gli altri comandi sono attivi come se si trattasse di una vera colonna della tabella. Il comando di colonna per Pro ne è un esempio.

"Pro" viene definito come *alias di colonna*, un altro nome da utilizzare quando si fa riferimento a una colonna. Come in ORACLE7.1, questi elementi possono essere specificati in una maniera leggermente diversa. È possibile utilizzare una clausola as per identificare gli alias di colonna nelle query. Così facendo l'istruzione select dell'esempio precedente verrebbe modificata in questo modo:

```
Quantita * Tasso AS Pro
```

L'utilizzo della clausola `as` aiuta a separare visivamente gli alias di colonna dalle colonne vere e proprie.

break on

Si osservi il punto 8 della Figura 5.3. Nel report della Figura 5.2, si noti come le transazioni per ciascun tipo di articolo siano raggruppate insieme sotto l'intestazione “Bene Venduto”. Questo effetto è stato ottenuto con la riga:

```
break on Articolo skip 2
```

e con la riga:

```
order by Articolo
```

del comando `select` verso la fine del file di avvio.

In SQLPLUS ciascuna riga viene considerata così come viene riportata da ORACLE e viene tenuta traccia del valore che appare in `Articolo`. Per le prime quattro righe, questo valore è “MANZO”, per cui la riga viene visualizzata da SQLPLUS così com’è.

Nella quinta riga il valore di `Articolo` cambia da “MANZO” a “STIVALI PER CAVALLI”. Vengono applicate da SQLPLUS le istruzioni relative all’interruzione: nel caso in cui l’articolo cambia, la visualizzazione normale riga dopo riga deve essere modificata saltando due righe.

Si può infatti osservare che sono presenti due righe tra le sezioni di `Articolo` nel report. Se gli articoli non fossero raggruppati per mezzo del comando `order by`, non avrebbe senso saltare due righe ogni volta che cambia l’articolo impostando il comando `break on`.

Ecco il motivo per il quale il comando `break on` e la clausola `order by` devono essere impiegati in maniera coordinata.

Si può anche osservare che la voce “MANZO” viene stampata soltanto sulla prima riga della sezione, come anche “STIVALI PER CAVALLI”, “BURRO”, “VITELLO” e “LATTE”. Questo serve per eliminare la ripetizione di ciascun articolo per ogni riga di ogni sezione, che risulta visivamente poco piacevole. Se lo si desidera, è invece possibile visualizzare ogni articolo in tutte le righe della sezione modificando il comando `break on` in questo modo:

```
break on Articolo duplicate skip 2
```

Nella visualizzazione del report della Figura 5.2 si può osservare che non viene indicato il totale complessivo. Per ottenere il totale complessivo di un report occorre aggiungere un’interruzione utilizzando il comando `break on report`. È necessario prestare attenzione quando si aggiungono delle interruzioni, poiché devono essere create con un unico comando: se si inseriscono due comandi `break on` consecutivi, le istruzioni del primo vengono ridefinite dal secondo.

Per ottenere il totale complessivo di questo report, occorre modificare in questo modo il comando `break on`:

```
break on Articolo duplicate skip 2 on report
```

compute sum

I totali calcolati per ciascuna sezione del report sono stati prodotti con il comando compute sum evidenziato al punto 9. Questo comando funziona sempre congiuntamente al comando break on e i totali calcolati sono sempre riferiti alla sezione specificata con tale comando. Può quindi essere opportuno considerare questi due comandi correlati come un blocco unico:

```
break on Articolo skip 2
compute sum of Pro on Articolo
```

In altre parole, in questo modo viene indicato a ORACLE di effettuare la somma delle proroghe di ciascun articolo. Viene quindi calcolata la somma prima per MANZO, poi per STIVALI PER CAVALLI, BURRO, VITELLO, e LATTE. Ogni volta SQLPLUS rileva un nuovo articolo, viene calcolato e visualizzato il totale per gli articoli precedenti. Con il comando compute sum viene anche visualizzata una riga di asterischi sotto la colonna che viene utilizzata in quel momento da break on e viene visualizzata la parola "sum" al di sotto. Per i report in cui è necessario aggiungere molte colonne, viene utilizzato un comando compute sum separato per ciascun totale. È anche possibile avere molti tipi diversi di interruzioni in report lunghi (per Articolo, Persona e Data, ad esempio) con comandi compute sum correlati. Si consulti il Capitolo 13 per ulteriori dettagli.

Si può utilizzare il 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 (ad esempio, indirizzi con break on per Citta), tuttavia non è vero il contrario.

NOTA *Ogni comando compute sum deve avere un comando break on che lo guida e la parte "on" di ognuno dei comandi deve corrispondere (come nel caso di "on Articolo" nel comando "compute sum of Pro on Articolo").*

Ecco i principi fondamentali:

- per ogni comando break on deve esistere un comando order by correlato;
- per ogni comando compute sum deve esistere un comando break on correlato.

È facile comprendere, ma è altrettanto facile dimenticare qualche pezzo.

set linesize

I tre comandi al punto 10 della Figura 5.3 controllano le dimensioni generali del report. Con set linesize si impone il numero massimo di caratteri che possono comparire in un'unica riga. Per documenti delle dimensioni di una lettera, questo numero è generalmente intorno a 70 o 80, a meno che la stampante utilizzata 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

nella riga successiva e ammassate una sotto l'altra. In realtà si può utilizzare questa disposizione ottenendo un buon effetto quando è necessario presentare numerosi dati. Nel Capitolo 13 viene illustrato un esempio di questo tipo.

Il comando `linesize` si utilizza anche per determinare il punto in cui deve essere centrato il titolo `ttitle` e quello in cui collocare data e numerazione di pagina. Entrambi questi dati compaiono nella prima riga superiore 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` viene impostato il numero totale delle righe di ogni pagina, compresi i titoli `ttitle`, `btitle`, le intestazioni di colonna e qualsiasi eventuale riga vuota. Per il formato da lettera o quello generalmente utilizzato per i computer, il numero impostato è solitamente 66 (6 righe 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”, poiché in realtà con questo comando vengono visualizzate righe vuote prima della prima riga superiore (quella in cui compaiono data e numerazione di pagina) di ciascuna pagina del report.

Si tratta di una funzione utile sia per modificare la posizione di report che vengono stampati su pagine singole con una stampante laser, sia per saltare le perforazioni tra le pagine della carta da computer.

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

Perciò, se si digita:

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

SQLPLUS produce un report che inizia con 9 righe vuote, seguite da 57 righe di informazioni (contando dalla data fino al `btitle`). Se si aumentano le dimensioni impostate con `newpage`, SQLPLUS visualizza meno righe di informazioni su ciascuna pagina, produce più pagine insieme.

Tutto ciò è facilmente comprensibile, ma che cosa è stato impostato con i comandi visualizzati al punto 10 della Figura 5.3?

```
set pagesize 50  
set newpage 0
```

Si tratta di dimensioni insolite per la pagina di un report. L'istruzione di immettere zero righe vuote tra le pagine è stata impostata da SQLPLUS? No. Il numero zero dopo newpage viene attivato per 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, in cui inizia la stampa del report.

La combinazione dei due comandi set pagesize 50 e set newpage 0 ha per effetto la produzione di un report il cui corpo di informazioni è esattamente di 50 righe, con un carattere di inizio pagina all'inizio di ogni pagina. Si tratta di una modalità più semplice e più facile per controllare la stampa delle pagine, rispetto a quella di destreggiarsi tra righe vuote e numero di righe per pagina. Come per ORACLE8, è possibile utilizzare il comando set newpage none, con il risultato di non avere nessuna riga vuota e nessun carattere di inizio pagina tra le pagine del report.

spool

Ai primordi dell'era dei computer, la memorizzazione dei file veniva fatta per la maggior parte su bobine di filo magnetico o nastro. Scrivere informazioni su un file ed eseguire lo spooling erano virtualmente la stessa cosa. Il termine "spool" è sopravvissuto ed è ora in genere riferito a qualsiasi processo di spostamento di informazioni da un luogo a un altro. In SQLPLUS,

```
spool activity.lst
```

indica a SQL di prendere tutte le istruzioni da SQLPLUS e trascriverle nel file ACTIVITY.LST. Una volta inserito questo comando, l'operazione viene effettuata finché non si digita il comando opposto, ovvero:

```
spool off
```

Ciò significa, ad esempio, che si potrebbe digitare:

```
spool work.fil
```

e quindi impostare 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.

Ogni prompt prodotto da SQLPLUS, ogni messaggio di errore, qualsiasi cosa compaia sullo schermo del computer durante l'operazione di spooling verrebbe trasferito nel file WORK.FIL. Non è possibile impostare discriminanti. Tutto viene registrato dal momento in cui si utilizza il comando spool al momento in cui si utilizza spool off, il che ci riporta al report di Talbot, al punto 11 della Figura 5.3:

```
spool activity.lst
```

La stringa appena riportata viene correttamente collocata come comando immediatamente precedente alla clausola select e il comando spool off segue immediatamente.

Se l'istruzione spool activity.lst fosse stata impostata in una posizione precedente, i comandi SQLPLUS impostati successivamente sarebbero finiti nella prima pagina del file del report; viceversa, vengono trascritti nel file ACTIVITY.LST, come illustrato nella Figura 5.2: i risultati delle 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 maniera nitida.

```
/* */
```

Nel punto 12 della Figura 5.3 viene mostrato come può essere incorporato un commento in un'istruzione SQL. Si tratta di un metodo e di un tipo di utilizzo differente dal comando remark discusso in precedenza. Il termine remark (o rem) deve apparire all'inizio della riga e funziona soltanto per la riga in cui compare. Inoltre, in un'istruzione SQL a righe multiple non è consentito inserire un commento utilizzando remark. Quindi, l'istruzione:

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

non è corretto. Non funziona e viene visualizzato un messaggio di errore. Tuttavia, è possibile incorporare un'osservazione in un comando SQL seguendo il metodo illustrato al punto 12, ovvero 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 sia inserita da questo punto in poi, anche se prosegue con molte parole per diverse righe, viene considerata come un commento fino all'inserimento del segno */, che indica la fine del commento in questione. Per segnalare l'inizio di un commento si possono anche utilizzare i caratteri "--" (due trattini). In questo caso, la fine della riga indica la fine del commento. Questo tipo di commento funziona esattamente come remark, tranne per il fatto che viene utilizzato il segno -- (due trattini) invece di rem.

All'interno della query contrassegnata dal punto 12, la funzione TO_DATE viene utilizzata per accertarsi che le date vengano impostate nella modalità del ventesimo secolo (il 1900). Se il valore del secolo non fosse stato precisato, sarebbe stato presunto da ORACLE, e poiché le date sono così lontane nel passato, avrebbe potuto essere selezionato un valore di secolo sbagliato per i dati inseriti. Le modalità di gestione dei valori di secolo e le possibilità di formattazione della funzione TO_DATE sono illustrati nel Capitolo 8.

Qualche chiarimento sulle intestazioni di colonna

È possibile che la differenza di denominazione che si nota in:

Quantita * Tasso Pro

e nella nuova intestazione assegnata alla colonna Articolo in:

column Articolo heading 'Bene!Venduto'

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

compute sum of Pro on Articolo

Perché non si utilizza “compute sum of Pro on ‘Bene!Venduto’”? Perché con i comandi SQLPLUS si fa riferimento soltanto alle colonne che compaiono effettivamente nel comando di selezione. Ogni comando è riferito a una colonna citata nella clausola select. Sia break on, sia compute sono riferiti esclusivamente alle colonne riportate in questa clausola. L'unico motivo per cui un comando column o compute possa essere riferito alla colonna Pro è che questa denominazione sia anch'essa presente nella clausola select. La rinomina di “Quantita * Tasso” come “Pro” è stata effettuata da SQL, non da SQLPLUS.

5.2 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, digitando ciascuno dei comandi ritenuti necessari e quindi eseguendolo in SQLPLUS per verificare che sia corretto. Tuttavia, quando si crea un report per la prima volta, risulta spesso più semplice operare interattivamente con SQLPLUS, modificando i formati di colonna, le query SQL, i titoli e i totali, finché inizia a prendere forma ciò che si ha in mente.

Editor della riga di comando

Quando si digita un comando SQL, ogni riga digitata viene ricordata da SQLPLUS così come è stata inserita e viene memorizzata nel buffer di SQL (uno strano nome per indicare la cartella Appunti di un computer in cui vengono memorizzati i comandi SQL).

Si supponga di digitare questa query:

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

Viene visualizzata la seguente risposta:

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

Se ci si accorge di aver scritto male “Argomento”, non è necessario ridigitare l’intera query. Si può interrogare l’editor della riga di comandi per effettuare la verifica. Innanzitutto, si chiede di visualizzare la query con il comando:

```
list
```

Viene immediatamente visualizzata la risposta:

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

Si osservi che tutte e tre le righe della query sono state visualizzate da SQL-PLUS e sono state numerate. È anche stato inserito un asterisco alla riga 3, che indica la riga che è possibile modificare. Tuttavia la riga da modificare è la riga 1, per cui occorre digitare:

```
list 1
```

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

Viene visualizzata la riga 1 come riga corrente. È ora possibile modificarla digitando:

```
change /Argometno/Argomento
```

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

Si può verificare nuovamente l’intera query con il comando precedente:

```
list
```

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

Se la verifica è positiva, si digita una barra singola dopo il prompt. Questo segno non fa parte del comando change e non è un’indicazione per l’editor; segnala a SQLPLUS che è possibile eseguire il comando SQL che si trova nel buffer.

È anche possibile inserire un blocco di comandi PL/SQL nel buffer (si consulti in proposito il Capitolo 22).

Tuttavia, poiché tutti questi comandi terminano con un punto e virgola, non è possibile eseguirli. In questo caso, al termine del blocco occorre digitare una barra per indicare la fine del messaggio.

Per ulteriori informazioni si consulti la voce BLOCK STRUCTURE nel Capitolo 37. Si può anche utilizzare un punto (.) per indicare la fine del messaggio, senza eseguire il blocco PL/SQL.

/

| 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 riga:

change \$Argometno\$Argomento

avrebbe ottenuto lo stesso risultato. Il primo carattere dopo la parola "change" viene considerato come quello prescelto per segnalare l'inizio e la fine del testo da correggere (questi segni vengono solitamente definiti *delimitatori*). È anche possibile cancellare la riga corrente, come mostrato qui 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
```

Con l'indicazione "del" viene cancellato ciò che si trova nella riga corrente. Come avviene in ORACLE7.2, per cancellare in una sola volta righe multiple si può impostare una serie di righe da cancellare con il comando **del**, specificando il primo e l'ultimo numero di riga del blocco desiderato. Per cancellare le righe da 3 a 7, occorre digitare **del 3 7**. In questo comando si deve lasciare uno spazio prima del numero della prima riga da cancellare (3) e un altro prima del numero dell'ultima riga da cancellare (7).

Se non si impostano gli spazi tra "3" e "7", SQLPLUS cerca di cancellare la riga 37. Per eliminare le righe da 2 fino alla fine del contenuto del buffer, occorre digitare **del 2 LAST**. Scrivendo completamente la parola "delete" (cancellare) vengono eliminate tutte le righe e la parola viene inserita alla riga 1. Si tratta di un comando che può causare molti problemi, quindi è consigliabile evitare di digitare l'intero comando "delete". Se lo scopo è quello di cancellare completamente il comando **select**, occorre digitare:

clear buffer

Se si desidera aggiungere qualcosa alla riga corrente, si può utilizzare il comando **append**:

```
list 1

1* select Argomento, Sezione, Pagina

append "Dove sta"

1* select Argomento, Sezione, Pagina "Dove sta"
```

con **append** il testo indicato viene collocato esattamente al termine della riga corrente, senza nessuno spazio. Per inserire uno spazio, come è stato fatto in questo caso, occorre digitare due spazi tra la parola **append** e il testo da aggiungere.

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

```
list

1 select Argomento, Sezione, Pagina "Dove sta"
2*   from GIORNALE

input where Sezione = 'A'

list

1 select Argomento, Sezione, Pagina "Dove sta"
2   from GIORNALE
3* where Sezione = 'A'
```

quindi impostare l'intestazione di colonna per la colonna DoveSiTrova:

```
column DoveSiTrova heading "DoveSiTrova"
```

e poi eseguire la query:

```
/
```

| ARGOMENTO | S | Dove sta |
|------------|---|----------|
| Notizie | A | 1 |
| Editoriali | A | 12 |

Per rivedere i comandi inseriti con l'editor della riga di comando si può utilizzare il comando **list**, si può modificare o cancellare la riga corrente (segnalata dall'asterisco) con i comandi **change** o **delete**, aggiungere qualcosa alla fine della riga corrente con **append**, inserire un'intera riga dopo la riga corrente con **input**. Una volta terminate le correzioni, occorre digitare una barra al prompt SQL> per eseguirle. Tutti questi comandi possono essere abbreviati tranne **del**, che deve essere digitato esattamente con le tre lettere.

Con l'editor della riga di comando è possibile modificare soltanto le istruzioni SQL, non i comandi di SQLPLUS. Se ad esempio è stato digitato **column Articolo format a18** e si desidera modificarlo in “**column Articolo format a20**”, occorre digitare

nuovamente l'intero comando (questo vale se si opera in modalità interattiva con SQLPLUS, mentre se i comandi sono salvati in un file, si possono naturalmente modificare con l'editor utilizzato).

Si osservi che, nella modalità interattiva, una volta iniziata la digitazione di un'istruzione SQL, è necessario completarla prima di poter inserire qualsiasi altro comando SQLPLUS, come formati di colonna con column o ttitle. Tutto ciò che segue il termine "select" viene considerato da SQLPLUS come parte del comando select finché non viene inserito un punto e virgola (;) al termine dell'ultima riga o una barra (/) all'inizio della riga che segue l'ultima dell'istruzione SQL.

Questa impostazione è corretta:

```
select * from REGISTRO;
```

```
select * from REGISTRO
/
```

Questa no:

```
select * from REGISTRO/
```

set pause

Nello sviluppo di un nuovo report, oppure quando si utilizza SQLPLUS per query semplici, 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 uno schermo pieno di informazioni per ciascuna pagina del report prodotta e la visualizzazione viene interrotta a ogni pagina per dare la possibilità di consultarla (appare la dicitura "Ancora. . ." nell'angolo inferiore sinistro) finché non viene premuto il tasto INVIO. Dopo l'impostazione dei vari titoli e intestazioni di colonna, si possono 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 ai comandi SQL sono ampie, o se semplicemente si desidera lavorare con il proprio editor, è necessario salvare le istruzioni SQL impostate fino a quel momento in modalità interattiva, trasferendole in un file:

```
save fred.sql
```

La risposta di SQLPLUS è:

```
Wrote file fred.sql
```

Le istruzioni SQL (ma non i comandi column, ttitle o altri comandi SQLPLUS) sono ora riportate in un file di nome FRED.SQL (un nome di propria scelta), che può essere manipolato con qualsiasi editor.

Se il file esiste già, occorre utilizzare l'opzione replace (abbreviata in repl) del comando save per salvare la nuova query in un file con quel nome. Per l'esempio citato la sintassi sarebbe:

```
save fred.sql repl
```

Modifica

Ogni utente ha il suo editor preferito. Alcuni dei più diffusi sono KEDIT, BRIEF, vi. Con SQLPLUS possono anche essere utilizzati programmi di elaborazione di testi come WordPerfect, Word e altri, ma soltanto se si salvano i file creati in formato ASCII (si consulti il manuale del proprio programma per le modalità specifiche necessarie per effettuare questa operazione). Gli editor sono comuni programmi e vengono generalmente eseguiti digitandone il nome al prompt del sistema operativo. Su un PC, ciò equivale a qualcosa di simile a questa stringa:

```
C:> mioeditor activity.sql
```

Nell'esempio, mioeditor è il nome dell'editor e ACTIVITY.SQL rappresenta il file che si desidera visualizzare (il file di avvio descritto in precedenza viene utilizzato in questo contesto soltanto come esempio, occorre naturalmente digitare il nome del file che si desidera effettivamente visualizzare). In altri tipi di computer non appare necessariamente il prompt C:>, ma in ogni caso qualcosa di equivalente. Se è possibile richiamare un editor con questa modalità nel proprio computer, quasi certamente è possibile farlo anche da SQLPLUS, tuttavia non si deve digitare il nome dell'editor, ma il comando edit:

```
SQL> edit activity.sql
```

Tuttavia, non è possibile effettuare questa operazione se non viene prima impostato il nome dell'editor che si intende utilizzare. Occorre quindi definire l'editor in SQLPLUS, in questo modo:

```
define _editor = "mioeditor"
```

Si osservi che viene inserito un trattino di sottolineatura prima della “e” di editor. Il nome dell'editor viene registrato da SQLPLUS (finché non si esce da SQLPLUS con il comando quit), e risulta possibile utilizzarlo ogni volta che si desidera. Nel paragrafo “Utilizzo di LOGIN.SQL per definire l'editor” sono riportate ulteriori indicazioni sulle modalità per attivare l'editor automaticamente.

Il comando host

Nell'eventualità improbabile che nessuno di questi metodi funzioni, se si desidera assolutamente utilizzare l'editor prescelto è possibile richiamarlo digitando:

```
host mioeditor activity.sql
```

Il comando host indica a SQLPLUS di tornare al sistema operativo per l'esecuzione del programma indicato (si può anche utilizzare il segno \$ al posto del termine "host") ed equivale a digitare mioeditor activity.sql al prompt C>. Per inciso, lo stesso comando host può essere utilizzato per eseguire pressoché tutti i comandi di sistema operativo da SQLPLUS, compresi dir, copy, move, erase, cls e altri ancora.

Come aggiungere comandi SQLPLUS

Dopo avere salvato le istruzioni SQL in un file come FRED.SQL, è possibile aggiungere qualsiasi comando SQLPLUS si desideri. In pratica si può procedere come per la creazione del file ACTIVITY.SQL della Figura 5.3. Una volta terminato di lavorare sul file, si può uscire dall'editor e tornare a SQLPLUS.

Utilizzo di LOGIN.SQL per definire l'editor

Per definire il proprio editor con SQLPLUS occorre inserire il comando define_editor in un file denominato:

`login.sql`

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; se non lo trova, ricerca il file nella home directory di ORACLE e se non lo trova nemmeno li 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 rappresenta un modo conveniente di impostare un proprio ambiente SQLPLUS, con tutte le opzioni preferite. L'uso avanzato del file LOGIN.SQL è trattato nel Capitolo 13, di seguito ne è riportato un esempio tipico:

```
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, è utilizzato per stabilire le impostazioni di default 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 37.

start

In SQLPLUS, si può verificare il lavoro effettuato sul testo eseguendo il file appena modificato:

```
start fred.sql
```

Tutti i comandi SQLPLUS e SQL presenti nel file vengono eseguiti, riga dopo riga, 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 verificare i risultati del proprio lavoro. Si tratta esattamente di ciò che è stato visualizzato nella Figura 5.2, il prodotto dell'avvio di activity.sql e dello spooling effettuato su di esso e riportato nel file activity.lst.

Per sviluppare un report, occorre eseguire ciclicamente le seguenti fasi.

1. Utilizzare SQLPLUS per creare interattivamente 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. Visualizzare il file test.sql utilizzando l'editor prescelto. Aggiungere al file i comandi column, break, compute, set e spool. Solitamente si effettua lo spooling in un file con estensione .lst, come test.lst. Uscire dall'editor.
3. Tornati in SQLPLUS, avviare il file test.sql con il comando start. Il risultato viene mostrato sullo schermo e riportato nel file test.lst. Questo file viene esaminato dall'editor.
4. Incorporare ogni modifica necessaria nel file test.sql ed eseguire nuovamente quest'ultimo.
5. Proseguire con la procedura finché il report è chiaro e corretto.

5.3 Controllo dell'ambiente SQLPLUS

In precedenza si è segnalato che con l'editor della riga di comando non è possibile modificare i comandi SQLPLUS, poiché si può agire soltanto sulle istruzioni SQL, ovvero le righe memorizzate nel buffer di SQL. Si è anche indicato che è possibile salvare le istruzioni SQL in un file in cui possono essere modificate utilizzando il proprio editor. Tutto ciò riguarda i comandi SQL, ma per i comandi SQLPLUS? Esiste un modo per sapere quali comandi SQLPLUS sono stati inseriti o modificati, e un modo per salvarli? Verificare i comandi è facile, salvarli è una procedura un po' più complessa, discussa nel Capitolo 13.

Se si desidera controllare come è stata definita una determinata colonna, occorre digitare:

```
column Articolo
```

senza altri elementi dopo il nome di colonna. SQLPLUS presenta un elenco delle istruzioni impartite per quella colonna, come mostrato di seguito:

```
column Articolo ON
heading 'Bene!Venduto' headsep '!'
format a18
truncate
```

Se si digita soltanto column, senza specificare il nome della colonna, vengono elencate le istruzioni relative a tutte le colonne:

```
column Pro ON
format 990.99

column Quantita ON
heading 'Quan'
format 9990

column TipoQuantita ON
heading 'Tipo'
format a8
truncate

column DataAzione ON
heading 'Data'

column Tasso ON
format 90.99

column Persona ON
heading 'Venduto a'
format a18
word_wrap
column Articolo ON
heading 'Bene!Venduto' headsep '!'
format a18
truncate
```

Le istruzioni impostate con i comandi ttitle, btitle, break e compute vengono visualizzate semplicemente digitando i nomi dei comandi, senza altri elementi. Vengono immediatamente visualizzate le impostazioni correnti. La prima riga di ciascun esempio è la richiesta impostata dall'utente; le righe seguenti sono le risposte di SQLPLUS:

```
ttitle
ttitle ON and is the following 56 characters:
Vendite per prodotto durante il 1901!Secondo semestre (lug-dic)

btitle
btitle ON and is the following 26 characters:
dal registro di G. B. Talbot

break
break on Articolo skip 2 nodup
```

```
compute
COMPUTE sum OF Pro ON Articolo
```

Per visualizzare le impostazioni (dette anche *parametri*) che seguono il comando set occorre utilizzare show:

```
show headsep
headsep "!" (hex 21)
```

```
show linesize
linesize 79
```

```
show pagesize
pagesize 50
```

```
show newpage
newpage 0
```

Si consulti il Capitolo 37 alle voci set e show per l'elenco completo dei parametri.

Le impostazioni di ttitle e btitle possono essere disabilitate utilizzando i comandi btitle off e ttitle off, come riportati nell'elenco seguente. A questi comandi non segue nessuna risposta.

```
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. Nella prima riga di ciascun esempio viene visualizzata l'istruzione dell'utente; nelle righe seguenti è illustrato il risponso di SQLPLUS:

```
clear columns
columns cleared
```

```
clear breaks
breaks cleared
```

```
clear computes
computes cleared
```

5.4 I fondamenti

Questo capitolo è molto denso, soprattutto per i principianti di SQLPLUS, ma riflettendoci i lettori concorderanno probabilmente sul fatto che i concetti presentati non sono veramente difficili. Se la Figura 5.3 poteva apparire complessa quando si è iniziata la lettura del capitolo, ora appare sicuramente diversa. Probabilmente tutte le righe risultano comprensibili e si riesce almeno ad avere un'idea di ciò che viene svolto e delle motivazioni.

Se lo si desidera, si può copiare il file activity.sql in un altro file con un nome diverso e iniziare a modificarlo a proprio piacimento, adattandolo alle tabelle impostate. Le strutture di qualsiasi report prodotto sono in definitiva molto simili.

Ci sono molti elementi nel file activity.sql, tuttavia ciascuno è composto da unità di costruzione semplici. Questo è l'approccio utilizzato per tutto il testo. ORACLE fornisce i mattoni da costruzione, in gran quantità, ma ogni unità considerata singolarmente risulta comprensibile e utile.

Nei capitoli precedenti è stato illustrato come selezionare i dati dal database, scegliendo determinate colonne e ignorandone altre, scegliendo determinate righe sulla base dei vincoli logici impostati, e come combinare insieme due tabelle per ottenere informazioni non disponibili con una sola di esse.

In questo capitolo è stato illustrato come impartire ordini comprensibili per SQLPLUS sulla formattazione e la produzione di pagine e intestazioni di report chiari.

Nei numerosi capitoli seguenti (in cui l'andatura è un po' più moderata) vengono fornite indicazioni su come modificare e formattare i dati riga per riga. Il livello di conoscenza e la familiarità con il programma dovrebbero crescere capitolo dopo capitolo; al termine della Parte 2 di questo libro, l'utente dovrebbe essere in grado di produrre velocemente report molto sofisticati, con grande vantaggio suo e dell'azienda per cui lavora.

• Capitolo 6

• **Ottenere e modificare informazioni di testo**

- 6.1 **I tipi di dati**
- 6.2 **Che cos'è una stringa?**
- 6.3 **Notazione**
- 6.4 **Concatenazione (||)**
- 6.5 **Come tagliare e incollare le stringhe**
- 6.6 **Le funzioni stringa order by e where with**
- 6.7 **Riepilogo**

In questo capitolo vengono introdotte le *funzioni stringa*, strumenti software che consentono di manipolare una stringa di lettere o di altri caratteri. Per un riferimento immediato sulle singole funzioni, si consulti il Capitolo 37 alle voci corrispondenti. In questo capitolo viene discussa in particolare la manipolazione delle stringhe di testo; per ricercare delle parole (comprese quelle provenienti da radici comuni e le corrispondenze imperfette), occorre utilizzare l'opzione ConText di ORACLE, descritta nei Capitoli 29 e 30.

Queste funzioni in generale producono due diversi tipi di risultati: creazione di nuovi oggetti partendo da oggetti vecchi e descrizioni degli oggetti.

In un ristorante, ad esempio, una delle “funzioni” di un cuoco principiante è quella di trasformare le arance in spremuta, la verdura in insalata e l’arrosto in fette di carne. Con questa funzione un oggetto viene preso (arance, verdura, arrosto) e modificato (in spremuta, insalata, fette). Il “risultato” della funzione è una versione modificata dell’oggetto originale.

L’addetto agli acquisti del ristorante conta le arance, le verdure, e gli arrosti. Con questa funzione gli oggetti non vengono modificati, ma descritti. Anche in questo caso viene prodotto un “risultato” sotto forma di numero: viene indicato quanti sono gli oggetti.

Anche con le funzioni di ORACLE si opera in queste due modalità. Con alcune funzioni viene prodotto un risultato che consiste in una modifica dell’informazione originaria, come la trasformazione da caratteri in minuscolo a caratteri in maiuscolo all’interno di una frase.

Con altre funzioni viene prodotto un risultato che fornisce indicazioni sulle informazioni, come la quantità di caratteri contenuti in una parola o in una frase. In effetti i due tipi di funzioni stringa di ORACLE non sono concettualmente più complessi degli esempi presentati.

NOTA Se si utilizza PL/SQL, è possibile creare proprie funzioni da utilizzare all'interno di blocchi PL/SQL con il comando *create function*. Si consulti il Capitolo 24 per ulteriori dettagli.

6.1 I tipi di dati

Le persone possono essere classificate secondo tipologie differenti sulla base di determinate caratteristiche (timidezza, irritabilità, estroversione, vivacità, stupidità) e così dati diversi possono essere classificati in “tipi di dati” sulla base di determinate caratteristiche.

I tipi di dati di ORACLE comprendono NUMBER, CHAR (abbreviazione di CHARACTER, carattere), DATE, VARCHAR2, LONG, RAW, LONG RAW, BLOB, CLOB e BFILE. Il significato dei primi tre è evidente, gli altri sono tipi di dati speciali che vengono presentati più avanti. Una spiegazione completa della funzione di ciascuno di questi tipi di dati è disponibile nel Capitolo 37 alla voce “Tipi di dati” o cercando i nomi dei singoli tipi di dati. Ciascuno di questi viene trattato in maniera dettagliata nei capitoli successivi. Come per le persone, alcuni dei “tipi” si trovano in abbondanza e alcuni altri sono piuttosto rari.

Se l'informazione è riferita al tipo di carattere (VARCHAR2 o CHAR), come lettere, segni di interpunkzione, numeri e spazi (detti anche caratteri *alfanumerici*), occorre utilizzare funzioni stringa per modificarla o per ottenere indicazioni su di essa. Con SQL di ORACLE sono disponibili molti di questi strumenti.

6.2 Che cos'è una stringa?

Si tratta di un concetto semplice: un insieme di cose allineate, come una fila di case, di pop-corn o di perle, di numeri, o una stringa di caratteri in una frase o in una scommeggiatura.

Le stringhe si trovano frequentemente nella gestione di informazioni. I nomi sono stringhe di caratteri, come in “Juan L'Heureaux”. I numeri di telefono sono stringhe di numeri, trattini, e talvolta parentesi, come in “(415) 555-2676”. Anche un numero puro, come “5443702”, può essere considerato sia come numero, sia come stringa di caratteri.

NOTA I tipi di dati rappresentati da numeri puri (con un punto decimale e un segno meno, se necessario) sono definiti “NUMBER” e non vengono generalmente considerati come stringhe. Un numero può essere utilizzato con modalità specifiche che non sono possibili per una stringa e viceversa.

Le stringhe che possono comprendere qualsiasi mescolanza di lettere, numeri, spazi e altri simboli (come quelli che si trovano sopra ai numeri nelle tastiere) vengono definite *stringhe di caratteri*, o semplicemente *caratteri* per brevità. In ORACLE questo tipo di dati viene abbreviato in “CHAR”.

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 all'impostazione di CHAR, vengono automaticamente inseriti degli spazi vuoti. Quando vengono confrontate le stringhe di tipo CHAR in ORACLE, queste vengono considerate della stessa lunghezza con l'inserimento di spazi vuoti. Ciò significa che, se si paragona la stringa "carattere " con "carattere" nelle colonne di tipo CHAR, le due stringhe vengono considerate uguali. Il tipo di dati VARCHAR2 rappresenta una stringa di lunghezza variabile; VARCHAR è equivalente al tipo di dati VARCHAR2, tuttavia questa equivalenza potrebbe essere modificata nelle versioni future di ORACLE, quindi è opportuno evitare di utilizzare il tipo di dati VARCHAR.

CHAR si utilizza per campi con stringhe di lunghezza prefissata e VARCHAR2 per tutti gli altri campi con stringhe di caratteri.

Le semplici funzioni stringa di ORACLE discusse in questo capitolo sono descritte nella Tabella 6.1. Nel Capitolo 15 vengono presentate funzioni di stringa più avanzate e il loro utilizzo.

Tabella 6.1 Semplici funzioni stringa di ORACLE.

| NOME DI FUNZIONE | UTILIZZO |
|------------------|---|
| | Unisce o concatena due stringhe. Il simbolo è detto barra verticale spezzata. |
| CONCAT | Concatena due stringhe (come). |
| INITCAP | Pone in maiuscolo la prima lettera di una parola o di una serie di parole. |
| INSTR | Trova la posizione di un carattere in una stringa. |
| LENGTH | Indica la lunghezza di una stringa. |
| LOWER | Converte ogni lettera di una stringa in minuscolo. |
| LPAD | Riempie una stringa fino a una certa lunghezza aggiungendo a sinistra una determinata serie di caratteri. |
| RPAD | Riempie una stringa fino a una certa lunghezza aggiungendo a destra una determinata serie di caratteri. |
| RTRIM | Elimina tutti i caratteri che rientrano in una serie specificata dalla parte destra di una stringa. |
| SOUNDEX | Trova parole che si pronunciano in modo simile a quelle specificate (in inglese). |
| SUBSTR | Ritaglia una porzione di una stringa. |
| UPPER | Converte ogni lettera di una stringa in maiuscolo. |

6.3 Notazione

Le funzioni vengono indicate con questo tipo di notazione:

FUNCTION(*stringa* [,*opzione*])

Il nome di funzione è espresso in caratteri maiuscoli. Gli elementi a cui è riferito (in genere una stringa) sono visualizzati in corsivo minuscolo. La parola *stringa* rappresenta effettivamente una stringa di caratteri oppure il nome di una colonna di caratteri in una tabella. Quando si utilizza una vera funzione stringa, una stringa di caratteri effettiva deve essere racchiusa tra apici singoli, mentre un nome di colonna non deve avere nessun apice.

In ogni funzione può comparire soltanto una coppia di parentesi. Tutti gli elementi coinvolti dalla funzione, così come eventuali istruzioni aggiuntive, devono essere racchiusi tra parentesi.

In alcune funzioni sono presenti delle opzioni, elementi che non sono sempre necessari perché la funzione operi come desiderato. Le opzioni sono sempre visualizzate tra parentesi quadre []. Nel paragrafo seguente, dedicato a LPAD e RPAD, viene riportato un esempio delle modalità di utilizzo delle opzioni.

Di seguito viene illustrato un esempio semplice 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 un’effettiva stringa di caratteri che deve essere convertita in caratteri minuscoli e viene visualizzata in corsivo minuscolo. Quindi:

`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. In ORACLE vengono utilizzati apici singoli per segnalare l’inizio e la fine di qualsiasi stringa letterale. Questa stringa nella 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 visualizzata con un comando select. Ad esempio:

`select Citta, LOWER(Citta), LOWER('Citta') from CLIMA;`

produrrebbe 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 |

In cima alla 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 co-

lonna e non di un elemento letterale. Nella terza colonna, CITTA è racchiusa tra apici singoli; ciò significa che la funzione LOWER deve agire letteralmente sulla parola ‘Citta’ (ovvero, la stringa formata dalle lettere c-i-t-t-a) e non sulla colonna che porta lo stesso nome.

6.4 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. Ad esempio:

```
select Citta||Paese from LOCAZIONE;
```

```
CITTA || PAESE
```

```
ATENEGRECIA
CHICAGOSTATI UNITI
CONAKRYGUINEA
LIMAPERU
MADRASINDIA
MANCHESTERINGILTERRA
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. Come si può fare se si desidera renderlo un po' più leggibile, magari con un elenco di città e paesi separati da una virgola e da uno spazio?

Occorre semplicemente 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 LOCAZIONE;
```

```
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 5 per un ripasso su questo argomento.

Per concatenare delle stringhe, si può anche utilizzare la funzione CONCAT. Ad esempio, la query:

```
select CONCAT(Citta, Paese) from LOCAZIONE;
```

è equivalente alla query:

```
select Citta||Paese from LOCAZIONE;
```

6.5 Come tagliare e incollare le stringhe

Nel seguito vengono illustrate varie di funzioni che spesso sono fonte di confusione per gli utenti: LPAD, RPAD, LTRIM, RTRIM, LENGTH, SUBSTR e INSTR. Hanno tutte uno scopo comune: *tagliare e incollare*.

Ciascuna di queste funzioni esegue una parte della procedura. Ad esempio, LENGTH consente di conoscere il numero di caratteri contenuti in una stringa. SUBSTR consente di ritagliare e utilizzare una sottostringa che inizia e termina in determinate posizioni all'interno della stringa.

Con INSTR si può 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.

Infine, LTRIM e RTRIM consentono di ritagliare dei caratteri al termine di una stringa. Tutte queste funzioni possono essere utilizzate in combinazione l'una con l'altra, come viene 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 insieme di caratteri. Si possono 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 per il lato sinistro della colonna.

Le sintassi per le funzioni RPAD e LPAD sono:

`RPAD(stringa, lunghezza [, 'set'])`

`LPAD(stringa, lunghezza [, 'set'])`

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, l'ampiezza della colonna) e *set* è l'insieme di caratteri utilizzati per il riempimento. L'insieme di caratteri deve essere racchiuso tra apici singoli. Con le parentesi quadre si segnala che l'indicazione dell'insieme di caratteri (e della virgola che lo precede) è facoltativa. Se non viene impostato questo parametro, il riempimento viene automaticamente effettuato utilizzando degli spazi. Talvolta questa opzione viene definita come opzione di *default*; ovvero, se non si indica quale insieme di caratteri deve essere utilizzato, vengono utilizzati per default gli spazi.

Molti utenti creano tavole con puntini come aiuto visivo per collegare un lato della pagina all'altro. Ecco come si può effettuare questa impostazione con la funzione RPAD:

```
select RPAD(Città,35,'.'), Temperatura from CLIMA;
```

| RPAD(CITTÀ,35,'.') | TEMPERATURA |
|--------------------|-------------|
| LIMA..... | 7 |
| PARIGI..... | 27 |
| MANCHESTER..... | 19 |
| ATENE..... | 36 |
| CHICAGO..... | 19 |
| SYDNEY..... | -5 |
| SPARTA..... | 23 |

Si osservi che cosa è accaduto. Con RPAD è stata presa ogni stringa di città, da Lima fino a Sparta, e sono stati concatenati dei puntini alla sua destra, aggiungendone esattamente la quantità sufficiente perché il risultato (nome della città più i puntini) avesse lunghezza pari a 35. Con la funzione di concatenazione (`||`) questo risultato non sarebbe stato possibile, ma sarebbe stato aggiunto lo stesso numero di puntini per ciascuna città, lasciando un margine irregolare sulla destra.

Con LPAD si ottiene lo stesso tipo di risultato, ma sul lato sinistro. Si supponga di voler riformattare città e temperature in modo che i nomi delle città siano allineati al margine destro:

```
select LPAD(Città,11), Temperatura from CLIMA;
```

| LPAD(CITTÀ, TEMPERATURA |
|-------------------------|
| LIMA 7 |
| PARIGI 27 |
| MANCHESTER 19 |
| ATENE 36 |
| CHICAGO 19 |
| SYDNEY -5 |
| SPARTA 23 |

LTRIM e RTRIM

Queste funzioni si utilizzano per rimuovere caratteri indesiderati a sinistra e a destra delle stringhe. Ad 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 iniziato i titoli con l'articolo, altri no. Come si fa per uniformare il tutto?

```
select Titolo from RIVISTA;
```

| | |
|------------------------------------|-------|
| TITOLO | ----- |
| THE BARBERS WHO SHAVE THEMSELVES. | |
| "HUNTING THOREAU IN NEW HAMPSHIRE" | |
| THE ETHNIC NEIGHBORHOOD | |
| RELATIONAL DESIGN AND ENTHALPY | |
| "INTERCONTINENTAL RELATIONS." | |

Le sintassi per le funzioni RTRIM e LTRIM sono:

```
RTRIM(stringa [,'set'])
```

```
LTRIM(stringa [,'set'])
```

stringa è il nome della colonna del database (o una stringa letterale), e *set* è l'insieme dei caratteri che si desidera eliminare. Se non viene specificato nessun insieme 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. In questo caso si eliminano innanzitutto le virgolette e i punti sulla destra, come mostrato di seguito:

```
select RTRIM(Titolo,'.."'`) FROM RIVISTA
```



Il comando precedente produce questo effetto:

```
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 i punti a destra dei titoli e le virgolette. L'insieme di caratteri che si desidera eliminare può avere la lunghezza desiderata. Viene controllato e ricontrallato il lato destro di ogni titolo finché non è stato rimosso ogni carattere della stringa impostata, ovvero finché è stato trovato il primo carattere della stringa che non era stato indicato nell'insieme di caratteri da eliminare.

Combinazione di due funzioni

E ora? Come sbarazzarsi delle virgolette sulla sinistra? Titolo è sepolto in mezzo alla funzione RTRIM. Ecco come combinare le funzioni.

Come si è già detto, quando è stato eseguito il comando select:

```
select Titolo from RIVISTA;
```

si è ottenuto come risultato il contenuto della colonna Titolo:

```
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 quello di 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"
```

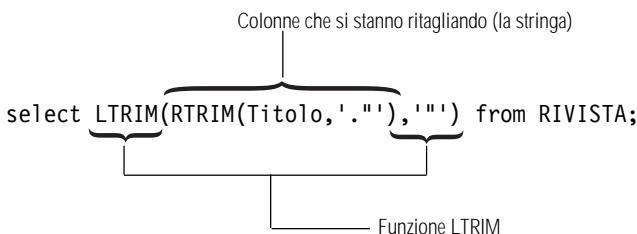
Perciò, se si suppone che RTRIM(Titolo,'.') sia esso stesso un nome di colonna, è possibile sostituirlo al termine *stringa* in:

```
LTRIM(stringa,'set')
```

Quindi occorre semplicemente digitare il comando select in questo modo:

```
select LTRIM(RTRIM(Titolo,'.'),'"') from RIVISTA;
```

Seguendo questo schema, il comando risulterà più chiaro:



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.

Può darsi che le cose non risultino tanto semplici. In effetti osservare una combinazione di funzioni per la prima volta può generare confusione, anche per un utente esperto di query. È difficile stabilire quali virgole 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 è quella di suddividere le funzioni in righe separate, almeno finché non funzionano nella maniera desiderata. Per SQLPLUS non ha nessuna importanza il punto in cui viene interrotto un comando 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 risulta ovvio e il comando funziona allo stesso modo anche se è digitato su quattro righe separate e con molti spazi. Gli spazi in più vengono semplicemente ignorati da SQLPLUS.

Si supponga di voler eliminare l'articolo “THE” dall'inizio dei titoli, con gli spazi che lo seguono (e naturalmente le virgolette che sono state rimosse in precedenza). Si potrebbe procedere così:

```
select LTRIM(RTRIM(Titolo,'."'),'"THE ')  
from RIVISTA;
```

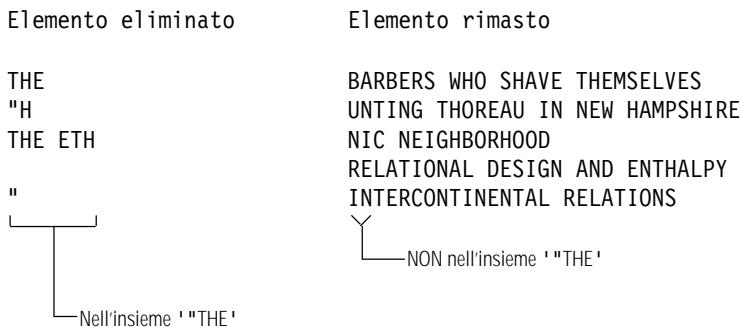
con questo risultato:

```
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. Come mai? Perché con LTRIM è stata eliminata qualsiasi virgoletta, T,

H, E o spazio. Non è stata cercata la parola “THE”, ma le sue lettere, e l’operazione non è stata interrotta dopo la prima volta in cui sono state trovate le lettere indicate, ma soltanto quando è stato trovato un carattere che non faceva parte dell’insieme impostato.



In altre parole, con tutte le seguenti:

```
'"THE'  
'HET'"  
'E"TH'  
'H"TE'  
'ET"H'
```

e molte altre combinazioni di queste lettere si ottiene lo stesso effetto, quando vengono impostate come insieme di un comando LTRIM o RTRIM. L’ordine delle lettere non ha nessun effetto sulle modalità operative della funzione. Si osservi invece che la differenza tra caratteri maiuscoli o minuscoli viene considerata, sia nelle lettere dell’insieme, sia nella stringa. Vengono rimosse soltanto le lettere esattamente corrispondenti.

Le funzioni LTRIM e RTRIM sono state create per eliminare qualsiasi carattere che fa parte di un insieme specificato alla sinistra o alla destra di una stringa, non hanno lo scopo di rimuovere parole.

Per ottenere questo risultato occorre utilizzare in maniera accorta le funzioni IN-STR, SUBSTR e anche DECODE, illustrate più avanti.

Un punto viene chiarito dall’esempio precedente: 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 della rivista avessero semplicemente evitato l’utilizzo di virgolette, punti e della parola “THE”.

Aggiunta di un’altra funzione

Si supponga di decidere di riempire il titolo ripulito con trattini e accenti circonflessi utilizzando la funzione RPAD, magari selezionando contemporaneamente il nome e il numero di pagina della 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 nella clausola select è quello di leggere dall'esterno verso l'interno sia a sinistra sia a destra, osservando (e anche contando) le coppie di parentesi.

LOWER, UPPER e INITCAP

Si tratta di altre tre funzioni correlate e molto semplici che spesso vengono utilizzate insieme. Con la funzione LOWER i caratteri di tutte le stringhe o di tutte le colonne vengono convertiti in lettere minuscole. Con la funzione UPPER si ottiene il risultato opposto, cioè 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.

Le sintassi per queste tre funzioni sono:

```
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 questo effetto:

| 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 osservi 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, la pulizia del titolo e l'introduzione del 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:

`LENGTH(stringa)`

Ad esempio:

```
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 |

Normalmente non è una funzione utile di per sé, ma può essere utilizzata come parte di un'altra funzione, per il calcolo dello spazio necessario in un report, o come parte di istruzioni con where od order by.

SUBSTR

È possibile utilizzare la funzione SUBSTR per ritagliare una parte di una stringa. La sintassi è:

`SUBSTR(stringa, inizio [,conta])`

Con questa notazione si indica a SQL di ritagliare una sottosezione della *stringa*, iniziando dalla posizione segnalata da *inizio* e proseguendo fino ai caratteri *conta*.

Se non viene specificata la posizione *conta*, l'operazione viene effettuata a partire da *inizio* e continuando fino alla fine della stringa. Ad esempio, con questo comando:

```
select SUBSTR(Nome,6,4) from RIVISTA;
```

si ottiene:

```
SUBS
-----
AND
FREE
OLOG
ISS
PY W
```

Si può osservare come opera la funzione. Sono state ritagliate le 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 quello di selezionare numeri telefonici da una rubrica personale. Ad 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 visualizzare soltanto i numeri di telefono con prefisso 415. Una soluzione sarebbe quella di ottenere una colonna separata di nome Prefisso. Pianificare con attenzione gli 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, si può 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 di default della funzione **SUBSTR**. **SUBSTR(Teléfono,5)** indica a SQL di ritagliare la sottostringa 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(Teléfono,5)
```

ha esattamente lo stesso effetto del seguente:

```
SUBSTR(Teléfono,5,8)
```

È possibile utilizzare tecniche di concatenazione e rinomina di colonna come quelle trattate nel Capitolo 5 per ottenere velocemente l'elenco dei numeri telefonici urbani, come illustrato di seguito:

```
select Cognome ||', ' ||Nome Nome, SUBSTR(Teléfono,5) Teléfono
  from INDIRIZZO
 where Teléfono 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 inserire una riga di puntini sul lato destro con la funzione **RPAD**, occorre aggiungere:

```
select RPAD(Cognome ||', ' ||Nome,25,'.') Nome,
       SUBSTR(Teléfono,5) Teléfono
  from INDIRIZZO
 where Teléfono 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 |

L'utilizzo di numeri negativi con la funzione **SUBSTR** non è documentato, ma è comunque possibile. Normalmente il valore specificato come posizione iniziale è calcolato relativamente all'inizio della stringa; quando invece si utilizza un numero

negativo come valore di posizione, è calcolato relativamente alla fine della stringa. Ad esempio, in:

```
SUBSTR(Teléfono,-4)
```

verrebbe utilizzata come punto iniziale la quarta posizione dalla fine del valore della colonna Teléfono. Poiché 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. Non deve assolutamente essere utilizzata con colonne impostate con tipo di dati CHAR. Le colonne 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 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. Utilizzando un valore di lunghezza negativo, si ottiene NULL.

INSTR

La funzione INSTR consente di effettuare ricerche semplici o complesse di un dato insieme di caratteri in una stringa, in maniera analoga alle funzioni LTRIM e RTRIM, tranne per il fatto che con INSTR non viene eliminato alcunché. Con questa funzione viene semplicemente visualizzata la posizione all'interno della stringa dell'elemento ricercato. 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 qualsiasi clausola, con l'eccezione di quelle con from. Naturalmente LIKE può essere utilizzato per ricerche complesse di modelli che sarebbe molto difficile, se non impossibile, effettuare utilizzando INSTR.

La sintassi la funzione INSTR è:

```
INSTR(stringa,set [,inizio [,ricorrenza ] ])
```

Con INSTR viene ricercato un determinato *set* di caratteri in una *stringa*. È possibile impostare due opzioni, che si trovano nella stringa di comando una all'interno

dell'altra; la prima è quella di default: l'insieme di caratteri viene ricercato partendo dalla posizione 1. Se si specifica la posizione da cui iniziare, vengono saltati tutti i caratteri precedenti il punto indicato e la ricerca comincia esattamente da quel punto.

La seconda opzione è la *ricorrenza*. Un *set* di caratteri può essere ricorrente in una stringa e talvolta si è interessati proprio alla ricorrenza. Come impostazione predefinita, INSTR ricerca la prima presenza del set di caratteri. Aggiungendo l'opzione *ricorrenza* con valore pari a 3, ad esempio, si può ottenere che vengano saltate le prime due ricorrenze e venga invece localizzata la terza.

Tutto ciò diventa più semplice da comprendere con l'ausilio di qualche esempio. Riprendendo la tabella degli articoli della rivista, ecco un elenco degli autori:

```
select Autore from RIVISTA;
```

| AUTORE |
|----------------------|
| BONHOEFFER, DIETRICH |
| CHESTERTON, G. K. |
| RUTH, GEORGE HERMAN |
| WHITEHEAD, ALFRED |
| CROOKES, WILLIAM |

| |
|---------------------|
| CHESTERTON, G. K. |
| RUTH, GEORGE HERMAN |
| WHITEHEAD, ALFRED |
| CROOKES, WILLIAM |

Per trovare la localizzazione della prima ricorrenza della lettera 'O', viene utilizzata la funzione INSTR senza nessuna opzione e con l'insieme di caratteri da ricerare indicato come 'O' (si osservi la presenza degli apici singoli, poiché si tratta di un elemento letterale):

```
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;
```

Se invece si desidera ricercare la seconda ricorrenza della lettera 'O', occorre utilizzare:

```
select Autore, INSTR(Autore,'O',1,2) from RIVISTA;
```

ottenendo questo risultato:

| 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, ovvero negativo, perché non è stata trovata una seconda ‘O’.

Per indicare a INSTR di cercare la seconda ricorrenza, occorre anche indicare la posizione in cui iniziare la ricerca (in questo caso la posizione 1). Il valore di default per *inizio* è 1, tuttavia per impostare l’opzione *ricorrenza* occorre impostare 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, 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 |

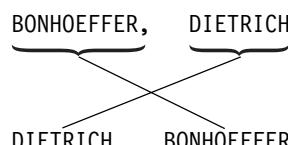
Questa possibilità può consentire diverse applicazioni utili. Nella tabella RIVISTA, ad esempio:

```
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 riformattare 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:



Utilizzando INSTR e SUBSTR, occorre cercare la posizione della virgola e quindi utilizzare il risultato per indicare a SUBSTR il punto da 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(AUTHO |
|----------------------|---------------------------|
| BONHOEFFER, DIETRICH | DIETRICH |
| CHESTERTON, G. K. | G. K. |
| RUTH, GEORGE HERMAN | GEORGE HERMAN |
| WHITEHEAD, ALFRED | ALFRED |
| CROOKES, WILLIAM | WILLIAM |

Si osservi la combinazione delle due operazioni ottenuta inserendo uno spazio tra di esse con la funzione di concatenazione e il veloce 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 |
|----------------------|---------------------|
| <hr/> | |
| BONHOEFFER, DIETRICH | DIETRICH BONHOEFFER |
| CHESTERTON, G. K. | G. K. CHESTERTON |
| RUTH, GEORGE HERMAN | GEORGE HERMAN RUTH |
| WHITEHEAD, ALFRED | ALFRED WHITEHEAD |
| CROOKES, WILLIAM | 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.

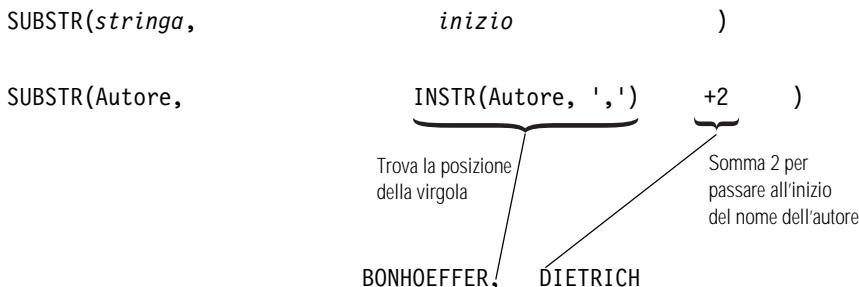
La prima parte della procedura è la seguente:

```
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:

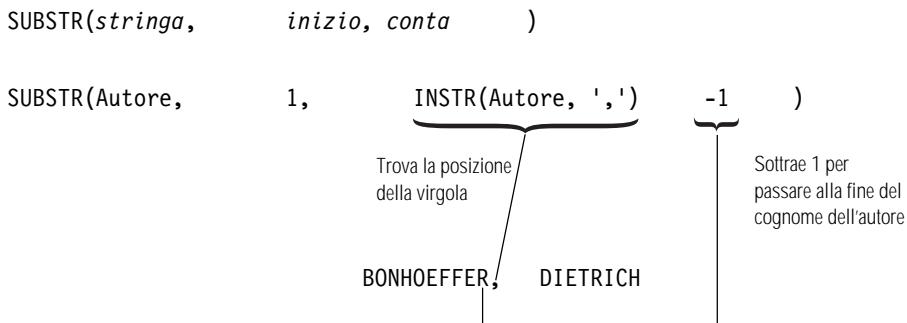
```
||' '||
```

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:



Nella quarta parte viene semplicemente rinominata la colonna:

"Per nome"

È stato possibile effettuare questa trasposizione soltanto perché in ogni record per 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.

Talvolta è difficile far rientrare i nomi nei formati standard. I cognomi possono avere dei prefissi (come "von" di "von Hagel") o suffissi (come "Jr.", "Sr.", e "III"). Utilizzando l'esempio precedente, il nome "Richards, Jr., Bob" sarebbe stato trasformato in "Jr., Bob Richards".

A causa della mancanza di una formattazione standard per i nomi, in molte applicazioni il nome e il cognome vengono memorizzati separatamente. I titoli (come "Dott.") sono solitamente collocati in un'altra colonna. Un'altra opzione quando si inseriscono i dati è quella di applicare un unico formato e utilizzare le funzioni SUBSTR e INSTR per manipolare i dati se necessario.

6.6 Le funzioni stringa order by e where with

Le funzioni 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 stese funzioni possono anche essere utilizzate in una clausola `order by`, come mostrato di seguito:

```
select Citta
  from CLIMA
 order by LENGTH(Citta);
```

| |
|------------|
| CITTA |
| ----- |
| LIMA |
| ATENE |
| PARIGI |
| SPARTA |
| SYDNEY |
| CHICAGO |
| MANCHESTER |

Quelli illustrati sono esempi semplici; si potrebbero impostare clausole molto più complesse. Ad esempio, si potrebbero cercare tutti gli autori con più di una ‘O’ nel nome utilizzando `INSTR` nella clausola `where`:

```
select Autore from RIVISTA
  where INSTR(Autore,'O',1,2) > 0;
```

| |
|----------------------|
| AUTORE |
| ----- |
| BONHOEFFER, DIETRICH |
| CROOKES, WILLIAM |

Questa istruzione cerca la seconda ricorrenza della lettera ‘O’ nel nome dell’autore. “`> 0`” è una tecnica logica: si ricordi che le funzioni generalmente producono due diversi tipi di risultati, uno con il quale vengono creati nuovi oggetti e l’altro con il quale vengono fornite informazioni su di essi.

Con la funzione `INSTR` vengono fornite informazioni su una stringa, in particolare la posizione dell’insieme di caratteri ricercato. In questo caso viene richiesto di localizzare la seconda ‘O’ nella stringa `Autore`. Il risultato è un numero superiore a zero per i nomi con almeno due ‘O’ e zero per quei nomi con una o meno ‘O’ (se non viene trovato niente il risultato è zero). Quindi, la semplice verifica della presenza di un risultato maggiore di zero conferma che la ricerca della seconda ‘O’ ha avuto successo. La clausola `where` in cui compare la funzione `INSTR` produce questo tipo di risultato:

```
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 o seguite da qualsiasi carattere. Si tratta di un esempio probabilmente più semplice da comprendere del precedente con `INSTR`.

Questo porta alla considerazione che esistono spesso diverse modalità per produrre lo stesso risultato in ORACLE. 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 stringa che viene utilizzata quasi esclusivamente in una clausola where: la funzione SOUNDEX. Con questa insolita funzione vengono trovate le parole che sono simili ad altre, anche 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 soltanto per termini di lingua inglese.

La sintassi della funzione SOUNDEX è il seguente:

`SOUNDEX(stringa)`

Ecco alcuni esempi di 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 singoli e viene ricercata la 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.

Uno scopo utile di questa funzione è quello di ripulire elenchi di indirizzi. 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 doppiioni 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 ottenere il risultato desiderato, occorre fare in modo che ogni cognome venga confrontato con tutti gli altri nella tabella.

Occorre unire la tabella INDIRIZZO con se stessa creando un alias della tabella, denominandola prima ‘a’ e poi ‘b’. Ora è come se le tabelle fossero due, a e b, con la stessa colonna 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:

```
select a.Cognome, a.Nome, a.Telefono
  from INDIRIZZO a, INDIRIZZO b
 where a.Cognome != b.Cognome
   and SOUNDEX(a.Cognome) = 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. Si consulti il Capitolo 29 per trovare esempi di questa e altre modalità di ricerca complesse.

Supporto di 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 in

caratteri analoghi. Si consultino le voci NLSSORT, NLS_INITCAP, NLS_LOWER, e NLS_UPPER del Capitolo 37.

6.7 Riepilogo

I dati possono essere di vari tipi, in primo luogo DATE, NUMBER, e CHARACTER. I dati di 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 stringa. In ORACLE sono disponibili due tipi di dati di 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, se il testo inserito risulta più corto della lunghezza di colonna definita.

Funzioni come RPAD, LPAD, LTRIM, RTRIM, LOWER, UPPER, INITCAP e SUBSTR modificano effettivamente il contenuto di una stringa o di una colonna prima della visualizzazione.

Funzioni come LENGTH, INSTR e SOUNDEX 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 da un database di ORACLE. Si tratta di procedimenti chiari, costituiti da fasi logiche semplici che possono essere combinate per ottenere risultati molto sofisticati.

Capitolo 7

Giocare con i numeri

- 7.1 Le tre classi delle funzioni numeriche
 - 7.2 Notazione
 - 7.3 Funzioni a valori singoli
 - 7.4 Funzioni di gruppo
 - 7.5 Funzioni di elenco
 - 7.6 Ricerca di righe con MAX o MIN
 - 7.7 Precedenza e parentesi
 - 7.8 Riepilogo

Qualsiasi cosa, particolarmente nel lavoro ma anche in altri aspetti della vita, viene misurata, esplicata 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 facilita 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.

7.1 Le tre classi delle 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 stringa (trattate nel Capitolo 6), con alcune di queste funzioni i valori vengono modificati, mentre con altre vengono semplicemente fornite delle informazioni sui valori. Le classi vengono distinte nel modo seguente.

Un valore singolo è un numero, come:

- un numero vero e proprio, ad esempio 544.3702;
 - una variabile in SQLPIU o in 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 vengono modificati mediante un calcolo.

Un *gruppo di valori* è costituito da tutti i numeri in una colonna di una serie di righe, come può essere il prezzo di chiusura per tutte le righe dei titoli di una tabella

AZIONE. Le funzioni di ORACLE per i gruppi di valori consentono di ottenere informazioni sull'intero gruppo, come il prezzo medio, ma non informazioni sui singoli valori.

Un *elenco di valori* è una serie di numeri che può comprendere:

- numeri veri e propri, come 1, 7.3, 22, 86;
- variabili in SQLPIU o in 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.

Nella Tabella 7.1 sono illustrate queste funzioni suddivise per classe; alcune compaiono in più di una classe, altre possono essere classificate sia come funzioni stringa, sia come funzioni numeriche, oppure vengono utilizzate per convertire i dati da una tipologia all'altra. Queste funzioni vengono discusse nel Capitolo 9 e sono elencate, suddivise per tipo, nella Tabella 9.1.

Tabella 7.1 Funzioni numeriche di ORACLE per classe. (*continua*)

FUNZIONI A VALORE SINGOLO

| FUNZIONE | DEFINIZIONE |
|------------------------------------|---|
| <i>valore1 + valore2</i> | Addizione. |
| <i>valore1 - valore2</i> | Sottrazione. |
| <i>valore1 * valore2</i> | Moltiplicazione. |
| <i>valore1 / valore2</i> | Divisione. |
| ABS(<i>valore</i>) | Valore assoluto. |
| CEIL(<i>valore</i>) | Il minimo intero maggiore o uguale a <i>valore</i> . |
| COS(<i>valore</i>) | Coseno di <i>valore</i> . |
| COSH(<i>valore</i>) | Coseno iperbolico di <i>valore</i> . |
| EXP(<i>valore</i>) | <i>e</i> elevato all'esponente <i>valore</i> . |
| FLOOR(<i>valore</i>) | Il massimo intero minore o uguale a <i>valore</i> . |
| LN(<i>valore</i>) | Logaritmo naturale di <i>valore</i> . |
| LOG(<i>valore</i>) | Logaritmo in base 10 di <i>valore</i> . |
| MOD(<i>valore, divisore</i>) | Modulo. |
| NVL(<i>valore, sostituto</i>) | Sostituto di <i>valore</i> se <i>valore</i> è NULL. |
| POWER(<i>valore, esponente</i>) | <i>valore</i> elevato a un esponente. |
| ROUND(<i>valore, precisione</i>) | Arrotondamento di <i>valore</i> a <i>precisione</i> . |
| SIGN(<i>valore</i>) | 1 se <i>valore</i> è positivo, -1 se negativo, 0 se zero. |
| SIN(<i>valore</i>) | Seno di <i>valore</i> . |
| SINH(<i>valore</i>) | Seno iperbolico di <i>valore</i> . |

Tabella 7.1 Funzioni numeriche di ORACLE per classe.**FUNZIONI A VALORE SINGOLO**

| FUNZIONE | DEFINIZIONE |
|------------------------------------|--|
| SQRT(<i>valore</i>) | Radice quadrata di <i>valore</i> . |
| TAN(<i>valore</i>) | Tangente di <i>valore</i> . |
| TANH(<i>valore</i>) | Tangente iperbolica di <i>valore</i> . |
| TRUNC(<i>valore, precisione</i>) | <i>Valore</i> troncato a <i>precisione</i> . |
| VSIZE(<i>valore</i>) | Dimensione di memorizzazione di <i>valore</i> in ORACLE. |

FUNZIONI DI GRUPPO

| FUNZIONE | DEFINIZIONE |
|---------------------------|--|
| AVG(<i>valore</i>) | Media di <i>valore</i> . |
| COUNT(<i>valore</i>) | Conteggio di righe per colonna. |
| MAX(<i>valore</i>) | Massimo di tutti i <i>valore</i> per gruppi di righe. |
| MIN(<i>valore</i>) | Minimo di tutti i <i>valore</i> per gruppi di righe. |
| STDDEV(<i>valore</i>) | Deviazione standard di tutti i valori per gruppi di righe. |
| SUM(<i>valore</i>) | Somma di tutti i <i>valore</i> per gruppi di righe. |
| VARIANCE(<i>valore</i>) | Varianza di tutti i <i>valore</i> per gruppi di righe. |

FUNZIONI DI ELENCO

| FUNZIONE | DEFINIZIONE |
|--|--|
| GREATEST(<i>valore1, valore2, ...</i>) | Il più grande <i>valore</i> di un elenco. |
| LEAST(<i>valore1, valore2, ...</i>) | Il più piccolo <i>valore</i> di un elenco. |

7.2 Notazione

Le funzioni sono riportate con questo tipo di notazione:

FUNCTION(*valore* [,*opzione*])

La funzione stessa viene indicata con caratteri maiuscoli. I valori e le opzioni sono in carattere corsivo minuscolo.

Quando appare il termine *valore* in questa modalità rappresenta un numero vero e proprio, il nome di una colonna numerica in una tabella, il risultato di un calcolo o una variabile.

Poiché in ORACLE non è consentito utilizzare i numeri come nomi di colonna, un numero vero e proprio non deve essere racchiuso tra apici singoli (che invece segnalano una stringa vera e propria in una funzione stringa). Anche i nomi di colonna non devono essere racchiusi tra apici.

In tutte le funzioni compare soltanto una coppia di parentesi. Tutto ciò su cui opera la funzione, ed eventuali istruzioni aggiuntive, devono essere racchiuse tra parentesi.

In alcune funzioni è possibile impostare delle opzioni, o delle parti che non sono indispensabili per il suo funzionamento, ma che possono offrire un maggiore controllo su di essa.

Le opzioni sono sempre visualizzate tra parentesi quadre []. Le parti indispensabili di una funzione sono sempre inserite prima delle parti facoltative.

7.3 Funzioni a valori singoli

La gran parte delle funzioni a valori singoli è decisamente semplice. In questo paragrafo vengono forniti alcuni brevi esempi e viene illustrato sia il risultato delle funzioni, sia le modalità di corrispondenza a colonne, righe ed elenchi. Dopo gli esempi, viene illustrata la procedura per combinare queste funzioni.

Una tabella denominata MATEMATICA è stata espressamente creata 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 (decimali inferiori) 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 operano sul valore di un numero.

A destra della colonna Nome sono presenti altre tre colonne: Sopra, che contiene soltanto i numeri sopra lo zero, Sotto, che contiene soltanto i numeri sotto lo zero, Vuoto, che contiene valori NULL.

NOTA *In ORACLE una colonna numerica può anche non avere nessun 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 viene illustrato più avanti.*

Non tutte le righe di questa tabella MATEMATICA sono necessarie per dimostrare l'utilizzo della maggior parte delle funzioni matematiche; negli esempi a seguire viene soprattutto utilizzata l'ultima riga, DECIMALE SUP.

Inoltre, si osservi che la formattazione di colonna è stata intenzionalmente applicata in modo da mostrare la precisione del calcolo. Poiché con SQLPLUS si può operare sulla formattazione dei numeri utilizzando il comando column, è importante non nascondere l'effetto delle funzioni matematiche di SQL con comandi column di SQLPLUS inappropriati, quindi sono state scelte accuratamente le opzioni di formattazione che consentano di visualizzare esattamente ciò che viene calcolato da SQL. Se si desidera conoscere i comandi SQL e SQLPLUS con i quali è stato prodotto l'effetto seguente, occorre fare riferimento al file MATEMATICA.sql (riportato nell'Appendice A) che li contiene. Nel Capitolo 13 vengono trattate le opzioni di formattazione dei numeri.

Addizione, sottrazione, moltiplicazione e divisione (+, -, * e /)

Ciascuna delle quattro funzioni aritmetiche fondamentali viene illustrata in questa query, 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 | PIU | MENO | PER | DIVISO |
|--------------|--------|---------|-------|---------|---------|--------------|---------|
| DECIMALE SUP | 66.666 | -77.777 | | -11.111 | 144.443 | -5185.081482 | .857143 |

NULL

Le stesse quattro operazioni sono eseguite nuovamente, questa volta utilizzando le colonne Sopra e Vuoto. Si osservi che qualsiasi operazione aritmetica che coinvolga un valore NULL ha come risultato un valore NULL. Le colonne di calcolo (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 | PIU | MENO | PER | DIVISO |
|--------------|--------|---------|-------|-----|------|-----|--------|
| DECIMALE SUP | 66.666 | -77.777 | | | | | |

Risulta qui evidente che un valore NULL non può essere utilizzato in un calcolo. NULL non coincide con zero: può essere considerato come un valore sconosciuto. Ad 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. Poiché 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 potrebbe essere NULL perché è sconosciuto (anche se in realtà esiste), mentre in altri casi risulta NULL semplicemente perché non esiste. Questo argomento viene esaminato in maniera più dettagliata più avanti in questo capitolo nel paragrafo “NULL nelle funzioni di gruppo”.

NVL, sostituzione di 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 sconosciuto, è possibile immaginare un'ipotesi ragionevole. Ad 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 per lui assolutamente impossibile 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 una visualizzazione delle informazioni fornite dai clienti.:

```
select Cliente, Peso from CONSEGNA;
```

| CLIENTE | PESO |
|---------------|------|
| JOHNSON TOOL | 59 |
| DAGG SOFTWARE | 27 |
| TULLY ANDOVER | |

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 in cui il pacco di un

cliente ha un peso sconosciuto, e quindi 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 al *valore*. *sostituto* può essere un numero vero e proprio, un'altra colonna o un calcolo. Nel caso del corriere, si potrebbe anche impostare un'unione di tabelle con il comando select in cui il valore di *sostituto* sia tratto da una vista con il calcolo del peso medio di tutti i pacchi di valore noto.

L'utilizzo della funzione NVL non è limitato ai numeri, ma si estende a tipi di dati CHAR, VARCHAR2, DATE e altri ancora, tuttavia *valore* e *sostituto* devono appartenere allo 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 di qualcosa. Ad 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 per la funzione ABS è la seguente:

ABS(*valore*)

Si osservino questi esempi:

$\text{ABS}(146) = 146$

$\text{ABS}(-30) = 30$

CEIL

La funzione CEIL (sta per “ceiling”, “soffitto”) produce semplicemente il minimo numero intero maggiore o uguale a un valore specificato. Occorre fare molta attenzione all'effetto di questa funzione su numeri negativi.

Nel listato seguente viene illustrata la sintassi della funzione CEIL e alcuni esempi del suo utilizzo:

CEIL(*valore*)

$\text{CEIL}(2) = 2$

$\text{CEIL}(1.3) = 2$

$\text{CEIL}(-2) = -2$

$\text{CEIL}(-2.3) = -2$

FLOOR

La funzione FLOOR (“pavimento”) è ovviamente l’opposto della funzione CEIL. La sintassi è la seguente:

FLOOR(*valore*)

| | | |
|-------------|---|----|
| FLOOR(2) | = | 2 |
| FLOOR(1.3) | = | 1 |
| FLOOR(-2) | = | -2 |
| 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 i caratteri, consentendo di verificare la trasmissione accurata di una stringa di numeri. Un esempio di questo utilizzo è illustrato nel Capitolo 16. Con la funzione MOD un valore viene diviso per un divisore e viene indicato il resto. MOD(23,6) = 5 significa dividere 23 per 6. La risposta è 3 con resto di 5, quindi il risultato del modulo è 5.

La sintassi per la funzione MOD è la seguente:

MOD(*valore,divisore*)

Sia *valore* sia *divisore* possono essere qualsiasi numero reale. Il valore di MOD è zero se *divisore* è zero o negativo. Si osservino gli esempi seguenti:

| | | |
|---------------|---|-------|
| MOD(100,10) | = | 0 |
| MOD(22,23) | = | 22 |
| MOD(10,3) | = | 1 |
| MOD(-30.23,7) | = | -2.23 |
| MOD(4.1,.3) | = | .2 |

Nel secondo esempio viene visualizzato l’effetto ottenuto con la funzione MOD ogni volta in cui il divisore è più grande del *dividendo* (il numero che deve essere diviso). Il dividendo viene indicato come risultato. Si osservi anche l’importante caso seguente, 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*)

| | | |
|------------|---|---|
| POWER(3,2) | = | 9 |
|------------|---|---|

| | | |
|------------------|---|-------------|
| POWER(3,3) | = | 27 |
| POWER(-77.777,2) | = | 6049.261729 |
| POWER(3,1.086) | = | 3.297264 |
| POWER(64,.5) | = | 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*)

SQRT(64) = 8
SQRT(66.666) = 8.16492
SQRT(4) = 2
SQRT(-9) produce il me

La radice quadrata di un numero negativo è un numero immaginario. In ORACLE non sono supportati i numeri immaginari, quindi viene visualizzato un messaggio di errore.

EXP, LN e LOG

Le funzioni EXP, LN e LOG vengono raramente utilizzate nei calcoli commerciali, ma sono molto comuni nel campo scientifico e tecnico. EXP è il valore e (2.71828183...) elevato a una data potenza; LN è il “logaritmo naturale”, o in base e . Le prime due funzioni rappresentano il reciproco l’uno 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*)

$$\begin{aligned} \text{EXP(3)} &= 20.085537 \\ \text{EXP(5)} &= 148.413159 \end{aligned}$$

$\text{LN}(\text{valore})$

$$\ln(3) = 1.098612$$

LOG(valore)

$\text{LOG}(\text{EXP}(1), 3) = 1.098612$
 $\text{LOG}(10, 100) = 2$

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 un dato numero di cifre decimali.

Le sintassi per le funzioni ROUND e TRUNC sono le seguenti:

```
ROUND(valore,precisione)
TRUNC(valore,precisione)
```

Nel listato seguente vengono applicate alcune proprietà degne di nota. Innanzitutto si osservi il primo semplice esempio di selezione dalla tabella MATEMATICA con il comando select. Vengono richieste due cifre di precisione (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 i numeri negativi dell'ultima riga sono stati arrotondati o troncati: 66.666 è stato arrotondato a un numero superiore, 66.67, ma il valore -77.777 è stato arrotondato a un numero inferiore, -77.78. 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 | ROUND | TRUNC | TRUNC |
|--------------|--------|---------|-----------|-----------|-----------|-----------|
| | | | (SOPRA,0) | (SOTTO,0) | (SOPRA,0) | (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 da 55.5 a 56. In questo modo vengono seguite le più comuni convenzioni di arrotondamento (secondo altre convenzioni, un numero viene arrotondato per eccesso soltanto se è il decimale 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 insieme con precisione con parametri negativi, sulle cifre a sinistra del punto decimale:

```
select Nome, Sopra, Sotto,  
       ROUND(Sopra,-1),  
       ROUND(Sotto,-1),  
       TRUNC(Sopra,-1),  
       TRUNC(Sotto,-1)  
  from MATEMATICA;
```

| NOME | | | ROUND | ROUND | TRUNC | TRUNC |
|--------------|--------|---------|------------|------------|------------|------------|
| | SOPRA | SOTTO | (SOPRA,-1) | (SOTTO,-1) | (SOPRA,-1) | (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 rappresenta il lato opposto 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 per la funzione **SIGN** è la seguente:

SIGN(*valore*)

Esempi: $\text{SIGN}(146) = 1$ Confrontare con: $\text{ABS}(146) = 146$
 $\text{SIGN}(-30) = -1$ $\text{ABS}(-30) = 30$

La funzione SIGN di 0 è 0:

$$\text{SIGN}(0)=0$$

La funzione SIGN viene spesso utilizzata insieme con la funzione DECODE. Questa funzione viene descritta nel Capitolo 16.

SIN, SINH, COS, COSH, TAN e TANH

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 trigonometrici 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

7.4 Funzioni di gruppo

Si tratta delle funzioni statistiche SUM, AVG, COUNT e di altre funzioni simili che forniscono informazioni su un gruppo di valori considerati come intero: l'età media di tutti gli amici nella tabella citata in precedenza, ad esempio, oppure il nome della persona più anziana del gruppo, o della 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, è comunque un'informazione che viene definita dalla relazione con il gruppo.

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 consideri ad esempio la funzione AVG. Si supponga di avere un elenco di 100 amici con le rispettive età. Se se ne prendono 20 a caso, e si calcola la media delle loro età, quanto varia il risultato rispetto a quello ottenuto prendendo un diverso elenco casuale di altre 20 persone, 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 media non è immune alla situazione in cui mancano di dati ed esistono casi in cui il calcolo così effettuato è significativamente diverso dal valore corretto (come quando i dati mancanti non sono distribuiti in maniera casuale), ma sono meno comuni.*

Si può provare a paragonare la relativa ininfluenza della mancanza di dati per la funzione AVG con quello che accade, ad 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 calcolata con la funzione AVG dei 20 nominativi, o la somma delle loro età calcolata con la funzione SUM? Si tratta di una questione completamente diversa dalla possibilità di valutare la somma di tutti e 1000 sulla base di 20 (in effetti, è esattamente la media calcolata con AVG dei 20, 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 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 sulla attendibilità dei risultati consente di definire con quale modalità le altre funzioni di gruppo rispondono alla presenza di dati NULL. Le funzioni STDDEV e VARIANCE sono misurazioni della tendenza centrale e anch'esse sono relativamente poco influenzabili dalla mancanza di dati (queste funzioni sono trattate più avanti in questo capitolo).

Le funzioni MAX e MIN consentono di misurare i valori estremi dei dati disponibili. Possono fluttuare ampiamente, mentre la media AVG riamane 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. Se si aggiunge un neonato, la media ritorna a 50 ma l'età minima è zero. Risulta chiaro che la presenza di valori NULL sconosciuti può influenzare profondamente le funzioni MAX, MIN e SUM, quindi occorre cautelarsi quando le si utilizza, in particolare se è presente una percentuale significativa di dati NULL.

È possibile creare funzioni in cui venga anche considerata la distribuzione dei dati e la quantità di valori NULL confrontata con la quantità di valori reali, e ottenerne 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 tale risultato è attendibile.

La funzione COUNT rappresenta un caso speciale; può essere utilizzata con valori NULL, ma fornisce sempre come risultato un numero, mai un valore NULL. La sintassi e l'utilizzo di questa funzione vengono illustrati tra breve, tuttavia, semplicemente per confrontarla con le altre funzioni di gruppo, è sufficiente segnalare che con questa funzione vengono contate tutte le righe non-NUL 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 viene richiesto di contare le righe della tabella di amici senza specificarne la colonna, il risultato visualizzato è 100. Un esempio di questa differenza 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 di temperatura fondamentali ordinati per città a mezzogiorno e a mezzanotte per 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 questo esempio, la tabella ha soltanto otto righe: i dati per le quattro giornate del 1993 per le città di San Francisco e Keene, nel New Hampshire. Si possono utilizzare le funzioni numeriche di ORACLE per analizzare queste città, la temperatura media, la variabilità della temperatura e così via, per il 1993. 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 | | VARCHAR2(13) |
| DATAAMP | | DATE |
| MEZZOGIORNO | | NUMBER |
| MEZZANOTTE | | NUMBER |

E contiene questi dati:

```
select * from COMFORT;
```

| CITTA | DATAAMP | MEZZOGIORNO | MEZZANOTTE |
|---------------|-----------|-------------|------------|
| SAN FRANCISCO | 21-MAR-93 | 16.9 | 5.7 |
| SAN FRANCISCO | 22-JUN-93 | 10.6 | 22.2 |
| SAN FRANCISCO | 23-SEP-93 | | 16.4 |
| SAN FRANCISCO | 22-DEC-93 | 11.4 | 4.3 |
| KEENE | 21-MAR-93 | 4.4 | -18 |
| KEENE | 22-JUN-93 | 29.5 | 19.3 |
| KEENE | 23-SEP-93 | 37.7 | 28.1 |
| KEENE | 22-DEC-93 | -22 | -18 |

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 ciò 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) |
|------------|--------------|------------|------------|------------|
| 12,97 | 3 | 16.9 | 10.6 | 38.9 |

AVG(Mezzogiorno) è la media delle tre temperature note, COUNT(Mezzogiorno) è il conteggio di quelle che non sono valori 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) |
|------------|
| 38.9 |

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, DataCamp, Mezzogiorno-Mezzanotte
from COMFORT
where Citta = 'KEENE';
```

| CITTA | DATA CAMP | MEZZOGIORNO-M |
|-------|-----------|---------------|
| KEENE | 21-MAR-93 | 22.4 |
| KEENE | 22-JUN-93 | 10.2 |
| KEENE | 23-SEP-93 | 9.6 |
| KEENE | 22-DEC-93 | -4 |

Con sole quattro righe da considerare, 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 -4 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 sbagliata, come mostrato di seguito:

```
select AVG(Mezzogiorno-Mezzanotte)
      from COMFORT
     where Citta = 'KEENE';
```

```
AVG(MEZZOGIORNO-MEZZANOTTE)
-----
         9,55
```

Per ottenere una risposta corretta, occorre impostare un valore assoluto, come in questo caso:

```
select AVG(ABS(Mezzogiorno-Mezzanotte))
      from COMFORT
     where Citta = 'KEENE';
```

```
AVG(ABS(MEZZOGIORNO-MEZZANOTTE))
-----
        11.55
```

Combinando le funzioni in questo modo si segue la stessa tecnica illustrata nel Capitolo 6, 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. Si osservi che è possibile inserire funzioni a valore singolo all'interno di funzioni di gruppo. Con le funzioni a valore singolo vengono calcolati i risultati per ciascuna riga e 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 di valori possono contenere funzioni a valore singolo (anche molte) al posto del loro valore.

Come si procede per combinare funzioni di gruppo? Innanzitutto non ha nessun senso annidarle in questo modo:

```
select SUM(AVG(Mezzogiorno)) from COMFORT;
```

Viene visualizzato il seguente messaggio di errore:

```
ERROR at line 1: ORA-0978: nested set 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 della clausola select, infatti il motivo per cui è stato prodotto il messaggio di errore è proprio l'assenza di questo comando. Questo argomento viene discusso nel Capitolo 10.

Può avere senso aggiungere, sottrarre, moltiplicare o dividere i risultati di due o più funzioni di gruppo. Ad esempio,

```
select MAX(Mezzogiorno) - MIN(Mezzogiorno)
      from COMFORT
     where Citta = 'SAN FRANCISCO';
```

```
MAX(MEZZOGIORNO)-MIN(MEZZANOTTE)
```

```
-----  
-12.6
```

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;
```

| CITTA | AVG(MEZZOGIORNO) | MAX(MEZZOGIORNO) | MIN(MEZZOGIORNO) | Varia |
|---------------|------------------|------------------|------------------|-------|
| KEENE | 12.4 | 37,7 | -22 | 59.7 |
| SAN FRANCISCO | 12.97 | 16.9 | 10.6 | 6.3 |

Questa query offre un buon esempio di come si possano 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 viene spiegata in maniera dettagliata nel Capitolo 10. In breve, in questo esempio aveva la funzione di vincolare le funzioni di gruppo a operare non sulla tabella completa, ma 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
```

```
-----  
37.7 12.4 -22 26.96 727
```

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 utilizzata per la funzione COUNT (la barra | significa “oppure”):

```
COUNT([DISTINCT | ALL] valore)
```

Ed ecco un esempio di applicazione:

```
select COUNT(DISTINCT Citta), COUNT(Citta), COUNT(*)
from COMFORT;
```

```
COUNT(DISTINCTCITTA) COUNT(CITTA) COUNT(*)
```

```
----- ----- -----
```

```
2 8 8
```

In questa query sono visualizzati alcuni risultati degni di nota. Innanzitutto, 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 Citta.

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, indipendentemente dal fatto che vi siano delle 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 otterrebbe questo risultato:

```
select AVG(DISTINCT Eta) Media, SUM(DISTINCT Eta) Totale
from COMPLEANNO;
```

MEDIA TOTALE

47 94

E questa, se si desiderasse conoscere l'età media dei propri amici, non sarebbe la risposta corretta. L'utilizzo 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 rispetto 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 nessun comando per utilizzare l'opzione ALL; se non si specifica DISTINCT, viene automaticamente utilizzata ALL.

7.5 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 * from COMFORT;
```

| CITTA | DATA | CAMP | MEZZOGIORNO | MEZZANOTTE |
|---------------|-----------|------|-------------|------------|
| SAN FRANCISCO | 21-MAR-93 | | 16.9 | 5.7 |
| SAN FRANCISCO | 22-JUN-93 | | 10.6 | 22.2 |
| SAN FRANCISCO | 23-SEP-93 | | | 16.4 |
| SAN FRANCISCO | 22-DEC-93 | | 11.4 | 4.3 |
| KEENE | 21-MAR-93 | | 4.4 | -18 |
| KEENE | 22-JUN-93 | | 29.5 | 19.3 |
| KEENE | 23-SEP-93 | | 37.7 | 28.1 |
| KEENE | 22-DEC-93 | | -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, di dicembre per Keene:

```
select Citta, DataCamp, GREATEST(Mezzanotte,Mezzogiorno) Alta,
       LEAST(Mezzanotte,Mezzogiorno) Bassa
  from COMFORT;
```

| CITTA | DATA | CAMP | Alta | Bassa |
|---------------|-----------|------|------|-------|
| SAN FRANCISCO | 21-MAR-93 | 16.9 | 5.7 | |
| SAN FRANCISCO | 22-JUN-93 | 22.2 | 10.6 | |
| SAN FRANCISCO | 23-SEP-93 | | | |

| SAN FRANCISCO | 22-DEC-93 | 11.4 | 4.3 | |
|---------------|-----------|------|------|--|
| KEENE | 21-MAR-93 | 4.4 | -18 | |
| KEENE | 22-JUN-93 | 29.5 | 19.3 | |
| KEENE | 23-SEP-93 | 37.7 | 28.1 | |
| KEENE | 22-DEC-93 | -22 | -18 | |

Sulla riga di settembre per San Francisco non viene riportato nessun valore, poiché con le funzioni GREATEST e LEAST non è possibile confrontare correttamente una temperatura di mezzanotte effettiva con una temperatura di mezzogiorno sconosciuta. In altri due casi la temperatura di mezzanotte era in realtà 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, colonne, numeri veri e propri, calcoli o combinazioni di altre colonne.

Le funzioni GREATEST e LEAST possono anche essere utilizzate con colonne di caratteri. Ad 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
```

7.6 Ricerca di righe con MAX o MIN

Quale città ha la più alta 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 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 seguente query non funzionerebbe:

```
select Citta, DataCamp, MAX(Mezzogiorno)
  from COMFORT;
```

Viene infatti verificata la colonna Citta e viene visualizzato questo messaggio di errore:

```
select Citta, DataCamp, MAX(Mezzogiorno)
*
ERROR at line 1: ORA-0937: not a single group set function
```

Questo messaggio è un po' ambiguo; significa che è stato individuato un difetto 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 innesca un conflitto.

Ecco come impostare la query:

```
select Citta, DataCamp, Mezzogiorno
  from COMFORT
 where Mezzogiorno = (select MAX(Mezzogiorno) from COMFORT);
```

| CITTA | DATA CAMP | MEZZOGIORNO |
|-------|-----------|-------------|
| KEENE | 23-SEP-93 | 37.7 |

Viene prodotta soltanto una riga, quindi si potrebbe pensare che la combinazione di una richiesta per le colonne Citta e DataCamp insieme con la funzione MAX per i valori di Mezzogiorno non sia contraddittoria come illustrato poc'anzi. Ma se invece fosse stata richiesta la temperatura minima?

```
select Citta, DataCamp, Mezzanotte
  from COMFORT
 where Mezzanotte = (select MIN(Mezzanotte) from COMFORT);
```

| CITTA | DATA CAMP | MEZZANOTTE |
|-------|-----------|------------|
| KEENE | 21-MAR-93 | -18 |
| KEENE | 22-DEC-93 | -18 |

Due righe! Più di una riga soddisfa la richiesta impostata con MIN, quindi si innescia un conflitto cercando di combinare una normale richiesta di colonna con una funzione di gruppo.

È anche possibile utilizzare due sottoquery, ciascuna con una funzione di gruppo (o una con una funzione di gruppo e l'altra no). Si supponga di voler conoscere la temperatura massima e minima a mezzogiorno durante l'anno:

```
select Citta, DataCamp, Mezzogiorno
  from COMFORT
 where Mezzogiorno = (select MAX(Mezzogiorno) from COMFORT)
   or Mezzogiorno = (select MIN(Mezzogiorno) from COMFORT);
```

| CITTA | DATA CAMP | MEZZOGIORNO |
|-------|-----------|-------------|
| KEENE | 23-SEP-93 | 37.7 |
| KEENE | 22-DEC-93 | -22 |

7.7 Precedenza e parentesi

Quando viene utilizzato più di un operatore aritmetico o logico in unico calcolo, quale viene eseguito prima? E l'ordine impostato conta? Si consideri l'esempio seguente:

```
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
```

Il motivo di questo risultato è la *precedenza*, che definisce l'ordine con il quale sono eseguiti i calcoli matematici, non soltanto in ORACLE ma anche nella matematica in genere. Le regole sono semplici: le parentesi hanno la precedenza assoluta, poi vengono considerate moltiplicazioni e divisioni, quindi addizioni e sottrazioni. Quando viene calcolata 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:

| | |
|-------------------------|-----------------------------|
| $2^*4/2^*3 = 12$ | (equivale a $(2^*4)/2^*3$) |
| $(2^*4)/(2^*3) = 1.333$ | |
| $4-2^*5 = -6$ | (equivale a $4 - (2^*5)$) |
| $(4-2)^*5 = 10$ | |

Anche AND e OR obbediscono alle regole sulla 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 delle difficoltà nel 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.

7.8 Riepilogo

Le funzioni a 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 a valore singolo modificano quasi sempre il valore della colonna a cui sono applicati. 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.

Ad esempio:

```
ROUND(99.308,-1) = 90
```

Il risultato della funzione ROUND è quello di modificare il valore 99.308 nel valore 90. Con le funzioni di elenco i valori non vengono modificati, ma vengono invece selezionati il più grande (con la funzione GREATEST) o il più piccolo (con la funzione LEAST) di una serie di valori in una riga. Sia le funzioni a valore singolo sia quelle di elenco non producono un risultato se uno dei valori considerati è NULL.

Entrambe queste funzioni possono essere utilizzate in tutti i casi in cui può essere utilizzata un'espressione, 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 o il minimo, il loro numero, o la deviazione standard e così via.

Nelle funzioni di gruppo i valori NULL vengono ignorati e questa caratteristica deve essere tenuta ben presente quando si impostano 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 indice delle informazioni da una o più tabelle. Nel Capitolo 10 vengono forniti ulteriori dettagli su queste caratteristiche aggiuntive.

Infine, il grado di precedenza matematica e logica influisce sull'ordine con il quale vengono valutate le query e ciò può avere un effetto disastroso sui risultati. È necessario prendere l'abitudine di utilizzare le parentesi per rendere esplicito e facile da comprendere l'ordine desiderato.

Capitolo 8

Le date

- 8.1 **Aritmetica delle date**
- 8.2 **ROUND e TRUNC in calcoli di date**
- 8.3 **Formattazione di TO_DATE e TO_CHAR**
- 8.4 **Date nelle clausole where**
- 8.5 **L'anno 2000**

Uno dei punti di forza di ORACLE è la capacità di memorizzare e calcolare le date e il numero di secondi, minuti, ore, giorni, mesi e anni tra una data e l'altra. ORACLE ha anche la straordinaria possibilità di formattare le date in qualsiasi modo, dal semplice "01-APR-98" a "1 aprile nell'anno 772 dal regno di Luigi IX". Molte di queste funzioni di formattazione e calcolo 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.*

8.1 Aritmetica delle date

DATE è un tipo di dati ORACLE, proprio come CHAR e NUMBER, e ha le sue specifiche proprietà. Il tipo di dati DATE è salvato in uno speciale formato interno ORACLE che comprende non soltanto il mese, il giorno e l'anno, ma anche l'ora, i minuti e i secondi. Il vantaggio di tutti questi dettagli dovrebbe essere ovvio; se si ha, ad esempio, un servizio di assistenza tecnica per i clienti, per ogni accesso al database ORACLE può automaticamente memorizzare la data e l'ora in una sola colonna.

È poi possibile formattare la colonna della data in un report in modo che mostri soltanto la data, o la data e l'ora, o la data, l'ora e i minuti, o la data, l'ora, i minuti e i secondi.

SQLPLUS e SQL riconoscono le colonne di DATE e capiscono che devono applicare le istruzioni dell'aritmetica delle date, non della matematica normale. Ad esempio, aggiungendo 1 a 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, dal momento che le date di ORACLE comprendono le ore, i minuti e i secondi, l'aritmetica delle date può diventare complessa, poiché si potrebbe ottenere che la differenza fra oggi e domani è pari a 0.516 giorni (questo verrà spiegato in seguito in questo capitolo).

NOTA *Gli esempi tratti dal libro mastro di Talbot riportano le date effettive delle transazioni come trovate lì, perciò molte partono dall'anno 1901. Negli script forniti nell'Appendice A, i valori delle date inseriti per questo libro mastro includono esplicitamente il valore del secolo. A seconda dei parametri del database (e della data corrente), un valore di '01' per l'anno potrebbe essere interpretato come 1901 oppure 2001. Gli esempi di questo capitolo, per evitare ambiguità sul secolo, utilizzano per l'anno tutte e quattro le cifre.*

SysDate

ORACLE fa ricorso al sistema operativo per la data e l'ora correnti e le rende disponibili attraverso una colonna speciale chiamata SysDate. SysDate può essere considerata come una funzione il cui risultato è sempre 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. Qui SysDate mostra la data odierna:

```
select SysDate from DUAL;
```

```
SYSDATE
-----
01-APR-98
```

La differenza tra due date

FESTA è una tabella con alcune delle importanti feste nazionali negli Stati Uniti per l'anno 1995:

```
select Festa, Dataeff, DataCelebrata from FESTA;
```

| FESTA | DATAEFF | DATACELEBRATA |
|-------------------------|-----------|---------------|
| NEW YEAR DAY | 01-JAN-95 | 01-JAN-95 |
| MARTIN LUTHER KING, JR. | 15-JAN-95 | 16-JAN-95 |
| LINCOLNS BIRTHDAY | 12-FEB-95 | 20-FEB-95 |
| WASHINGTONS BIRTHDAY | 22-FEB-95 | 20-FEB-95 |
| FAST DAY, NEW HAMPSHIRE | 22-FEB-95 | 22-FEB-95 |
| MEMORIAL DAY | 30-MAY-95 | 29-MAY-95 |
| INDEPENDENCE DAY | 04-JUL-95 | 04-JUL-95 |

| | | |
|--------------|-----------|-----------|
| LABOR DAY | 04-SEP-95 | 04-SEP-95 |
| COLUMBUS DAY | 08-OCT-95 | 09-OCT-95 |
| THANKSGIVING | 23-NOV-95 | 23-NOV-95 |

Si può facilmente trovare quali feste nel 1995 non sono state festeggiate nella data effettiva del loro anniversario, sottraendo DataCelebrata da Dataeff. Se la risposta non è zero, le due date sono diverse:

```
select Festa, Dataeff, DataCelebrata
      from Festa
     where DataCelebrata - Dataeff != 0;
```

| FESTA | DATAEFF | DATACELEBRATA |
|-------------------------|-----------|---------------|
| MARTIN LUTHER KING, JR. | 15-JAN-95 | 16-JAN-95 |
| LINCOLNS BIRTHDAY | 12-FEB-95 | 20-FEB-95 |
| WASHINGTONS BIRTHDAY | 22-FEB-95 | 20-FEB-95 |
| MEMORIAL DAY | 30-MAY-95 | 29-MAY-95 |
| COLUMBUS DAY | 08-OCT-95 | 09-OCT-95 |

La tabella DUAL per test e calcoli veloci

DUAL è una piccolissima tabella fornita da ORACLE, con una sola riga e una sola colonna:

```
describe DUAL
```

| Name | Null? | Type |
|-------|-------|---------|
| DUMMY | | CHAR(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. Ad 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.

Funzioni di data

- ADD_MONTHS(*conta,data*) somma *conta* mesi a *data*.
 - 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*.
 - MONTHS_BETWEEN(*data2,data1*) fornisce *data2-data1* in mesi (può essere un numero frazionario).
 - NEXT_DAY(*data,'giorno'*) fornisce la data del giorno successivo a *date*, dove ‘giorno’ è ‘Monday’, ‘Tuesday’ e così via.
 - NEW_TIME(*data,'questo', 'altro'*) fornisce la data (e l’ora) nel fuso orario *questo*, che deve essere sostituito da un’abbreviazione di tre lettere per indicare il fuso orario corrente. *altro* deve essere sostituito da un’abbreviazione di tre lettere per il fuso orario di cui si vuole conoscere la data e l’ora.

I fusi orari sono i seguenti:

| | |
|---------|--|
| AST/ADT | ora standard/durante il giorno dell'Atlantico |
| BST/BDT | ora standard/durante il giorno di Bering |
| CST/CDT | ora standard/durante il giorno della zona centrale |
| EST/EDT | ora standard/durante il giorno dell'Est |
| GMT | ora media di Greenwich |
| HST/HDT | ora standard/durante il giorno dell'Alaska-Hawaii |
| MST/MDT | ora standard/durante il giorno delle Montagne |
| NST | ora standard di Terranova |
| PST/PDT | ora standard/durante il giorno del Pacifico |
| YST/YDT | ora standard/durante il giorno dello Yukon |

- `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, si può consultare il paragrafo “ROUND” nel Capitolo 37.
 - `TRUNC(data,'formato')` senza un formato imposta una data a 12 A.M. (mezzanotte, l'inizio del giorno). Per l'utilizzo di un formato per l'arrotondamento, si può consultare il paragrafo “TRUNC” nel Capitolo 37.
 - `TO_CHAR(data,'formato')` riformatta la data secondo il formato specificato.*
 - `TO_DATE(stringa,'formato')` converte la data da un particolare formato ('formato') in una data ORACLE. Accetta anche un numero invece di una stringa, con determinati limiti. 'formato' è ristretto.*

*Si consulti il paragrafo “Formati delle date” più avanti in questo capitolo.

Aggiunta di mesi

Se il 22 febbraio è il “Fast Day” nel New Hampshire, forse sei mesi dopo potrebbe essere celebrato come “Giorno di festa”. Se è così, qual è la data? Basta semplicemente utilizzare la funzione ADD_MONTHS per sommare 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-95
```

Sottrazione di mesi

Se la prenotazione del posto per il picnic deve essere fatta almeno sei mesi prima del Columbus Day, qual è l'ultimo giorno utile per effettuarla? Occorre considerare la data di celebrazione per il Columbus Day e utilizzare ADD_MONTHS per aggiungere un numero negativo di sei mesi (che è equivalente 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
-----
08-APR-95
```

GREATEST e LEAST

Per le feste spostate alla domenica, 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(Dataeff, DataCelebrata) Primo,
       Dataeff, DataCelebrata
  from FESTA
 where Dataeff - DataCelebrata != 0;
```

| | |
|-------------------------|----------------|
| FESTA | DATA CELEBRATA |
| ----- | |
| MARTIN LUTHER KING, JR. | 04-SEP-95 |

• •

NEXT_DAY

NEXT_DAY calcola la data di un particolare giorno della settimana (domenica, lunedì, martedì, mercoledì, giovedì, venerdì o sabato) successivo alla data specificata. Ad esempio, si supponga 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-95 |
| 15-FEB-95 |
| 15-MAR-95 |
| 15-APR-95 |
| 15-MAY-95 |
| 15-JUN-95 |
| 15-JUL-95 |
| 15-AUG-95 |
| 15-SEP-95 |
| 15-OCT-95 |
| 15-NOV-95 |
| 15-DEC-95 |

Quali sono le date di paga reali?

```
column GIORNOPAGA heading "Giorno paga!"
```

```
select NEXT_DAY(DataCiclo,'FRIDAY') GIORNOPAGA
      from GIORNOPAGA;
```

Giono paga!

| |
|-----------|
| 20-JAN-95 |
| 17-FEB-95 |
| 17-MAR-95 |
| 21-APR-95 |
| 19-MAY-95 |
| 16-JUN-95 |
| 21-JUL-95 |
| 18-AUG-95 |
| 22-SEP-95 |
| 20-OCT-95 |
| 17-NOV-95 |
| 22-DEC-95 |

Questo risultato è quasi corretto, eccetto per settembre e dicembre, perché NEXT_DAY è la data del venerdì successivo a quella del giorno di paga. Dal momento che il 15 settembre e il 15 dicembre cadono di venerdì, si ottiene (erroneamente) il venerdì seguente. La versione corretta è questa:

```
column GIORNOPAGA heading "Giorno paga!"  
  
select NEXT_DAY(DataCiclo-1,'FRIDAY') GIORNOPAGA  
from GIORNOPAGA;
```

Giorno paga!

```
20-JAN-95  
17-FEB-95  
17-MAR-95  
21-APR-95  
19-MAY-95  
16-JUN-95  
21-JUL-95  
18-AUG-95  
15-SEP-95  
20-OCT-95  
17-NOV-95  
15-DEC-95
```

NEXT_DAY è in realtà una funzione del tipo “maggiore di”, infatti richiede la data maggiore di quella specificata e che cada di venerdì ('FRIDAY'). Per tenere conto anche delle date che cadono già di venerdì, occorre sottrarre 1 dalla data del periodo di paga. I giorni di paga sono quindi sempre i venerdì giusti (se si vuole, si può considerare che cosa succede se una delle date del periodo cade di sabato e si è impostato select in questo modo).

LAST_DAY

Questa funzione restituisce la data dell'ultimo giorno del mese. Se, ad esempio, le commissioni e i premi vengono pagati sempre l'ultimo giorno del mese, è possibile calcolare queste date per il 1995.

```
column FineMese heading "Fine mese"  
  
select LAST_DAY(DataCiclo) FineMese  
from GIORNOPAGA;
```

Fine mese

```
31-JAN-95  
28-FEB-95  
31-MAR-95  
30-APR-95  
31-MAY-95
```

30-JUN-95
31-JUL-95
31-AUG-95
30-SEP-95
31-OCT-95
30-NOV-95
31-DEC-95

MONTHS_BETWEEN

Si supponga che tutti gli amici di un gruppo abbiano fornito le loro età, ma che poi si sia trovato un file con le loro date di nascita. Si possono caricare le informazioni in una tabella COMPLEANNO e visualizzarle:

```
select * from COMPLEANNO;
```

| NOME | COGNOME | DATANASCITA | ETA |
|----------|---------|-------------|-----|
| GEORGE | SAND | 12-MAY-46 | 42 |
| ROBERT | JAMES | 23-AUG-37 | 52 |
| NANCY | LEE | 02-FEB-47 | 42 |
| VICTORIA | LYNN | 20-MAY-49 | 42 |
| FRANK | PILOT | 11-NOV-42 | 42 |

Si vede che tutti hanno mentito fornendo età sbagliate. Per trovare le età reali, occorre calcolare i mesi trascorsi tra le date di nascita e la data odierna e dividerli per 12, per ottenere gli anni:

```
select Nome, Cofnome, Datanascita, Eta,
       MONTHS_BETWEEN(SysDate,Datanascita)/12 "Età effettiva"
  from COMPLEANNO;
```

| NOME | COGNOME | DATANASCITA | ETA | Età effettiva |
|----------|---------|-------------|-----|---------------|
| GEORGE | SAND | 12-MAY-46 | 42 | 51.99 |
| ROBERT | JAMES | 23-AUG-37 | 52 | 60.71 |
| NANCY | LEE | 02-FEB-47 | 42 | 51.26 |
| VICTORIA | LYNN | 20-MAY-49 | 42 | 48.97 |
| FRANK | PILOT | 11-NOV-42 | 42 | 55.49 |

Combinazione di funzioni di data

Si supponga di essere stati assunti il 1 aprile 1998 in un nuovo posto di lavoro, con un salario iniziale inferiore alle aspettative, ma con la promessa di una revisione dopo sei mesi, il primo giorno del mese successivo. Qual è il giorno della revisione?

```
select SysDate Oggi,
       LAST_DAY(ADD_MONTHS(SysDate,6)) + 1 Revisione
  from DUAL;
```

```
OGGI      REVISIONE  
-----  
02-APR-98 01-NOV-98
```

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. Se si vuole poi calcolare quanti giorni mancano alla revisione, si deve semplicemente sottrarre la data attuale. Si noti l'utilizzo delle parentesi per assicurare l'ordine di calcolo appropriato:

```
select (LAST_DAY(ADD_MONTHS(SysDate,6))+ 1)-SysDate Attesa  
from DUAL;
```

```
ATTESA  
-----  
213
```

8.2 ROUND e TRUNC in calcoli di date

Questa è SysDate di oggi:

```
SYSDATE  
-----  
01-APR-98
```

All'inizio del capitolo è stato notato che ORACLE può sottrarre una data dall'altra, ad esempio ieri meno oggi, e produrre una risposta che non è un numero intero. È opportuno esaminare questa situazione:

```
select TO_DATE('02-APR-98')-SysDate from DUAL;  
  
TO_DATE('02-APR-98')-SYSDATE  
-----  
.516
```

La ragione del numero frazionario di giorni tra oggi e domani è che ORACLE nelle date ricorda 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 giorno, ORACLE fa due ipotesi che riguardano le date.

- A una data inserita come letterale, come '01-APR-98', 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 posta a 12 A.M. (mezzanotte).
- SysDate include 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 mezzo-

giorno. 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('02-APR-98')-ROUND(SysDate) from DUAL;
TO_DATE('02-APR-98')-ROUND(SYSDATE)
-----
1
```

ROUND, senza un formato specificato, 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.

8.3 Formattazione di TO_DATE e TO_CHAR

TO_DATE e TO_CHAR sono simili in quanto hanno potenti capacità di formattazione, ma in realtà sono tra loro complementari, dal momento che TO_DATE converte una stringa di caratteri o un numero in una data di ORACLE e TO_CHAR converte una data di ORACLE in una stringa di caratteri. Le sintassi di queste due funzioni sono le seguenti:

```
TO_CHAR(data[,'formato'[,'NLSparameters']])
TO_DATE(stringa[,'formato'[,'NLSparameters']])
```

date deve essere una colonna definita in ORACLE come tipo DATE. Non può essere una stringa, anche se è nel formato di default DD-MON-YY. L'unico modo di utilizzare una stringa al posto di date nella funzione TO_CHAR è quello di includerla in una funzione TO_DATE.

string è una stringa letterale, un numero letterale o una colonna di un database che contiene una stringa o un numero. In tutti i casi, tranne uno, il formato deve corrispondere a quello descritto da *formato*. Solo se una stringa è nel formato di default, *formato* può essere tralasciato. Il valore di default è DD-MON-YY, ma può essere cambiato con:

```
alter session set NLS_DATE_FORMAT
```

per una data sessione SQL o con il parametro NLS_DATE_FORMAT di init.ora.

Per *formato* si può scegliere tra più di 40 opzioni, che possono essere combinate in un numero praticamente infinito di modi. Queste opzioni con le rispettive spiegazioni sono elencate nel paragrafo “Formati delle date”. Una volta compreso il metodo di base di utilizzo delle opzioni, applicarle è semplice.

NLSparameters è una stringa che imposta l'opzione NLS_DATE_LANGUAGE a un particolare linguaggio, invece di utilizzare il linguaggio della sessione SQL corrente,

soltamente non la si utilizza spesso. ORACLE restituisce i nomi del giorno e del mese nel linguaggio impostato per la sessione con alter session.

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 SQL-PLUS grande circa 100 caratteri. Rinominando la colonna per renderla più leggibile e impostando il formato a 30 caratteri si ottiene una visualizzazione nello stile americano:

```
column Formattato format a30 word_wrapped
select Datanascita, TO_CHAR(Datanascita,'MM/DD/YY') Formattata
  from COMPLEANNO
 where Nome = 'VICTORIA';

DATANASCITA FORMATTATA
-----
20-MAY-49 05/20/49
```

Datanascita mostra il formato di default della data in ORACLE: DD-MON-YY per giorno del mese, trattino, tre lettere di abbreviazione per il mese, trattino, ultime due cifre dell'anno. La funzione TO_CHAR in select è banale. MM, DD e YY nell'istruzione TO_CHAR sono simboli chiave per la formattazione della data.

Le barre (/) sono soltanto segni di interpunkzione; ORACLE ne accetta anche altri, non hanno un senso particolare. Ad 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 racchiuso tra doppi apici:

```
select Datanascita, TO_CHAR(Datanascita,'Month, DDth "in, um,"YyyY')
Formattata
  from COMPLEANNO ;

DATANASCITA FORMATTATA
-----
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 qui varie conseguenze del formato. La parola intera Month comunica a ORACLE di utilizzare nella visualizzazione il nome intero del mese. Poiché è stata specificata con la prima lettera maiuscola e le altre minuscole,

ogni mese nel risultato viene formattato esattamente allo stesso modo. Le opzioni per il mese sono le seguenti:

- Month produce August
- month produce august
- Mon produce Aug
- mon produce aug

Il giorno del mese viene ottenuto con DD. Un suffisso th aggiunto a DD comunica a ORACLE di utilizzare insieme al numero i suffissi ordinali inglesi come "TH", "RD" e "ND". In questo caso, anche i suffissi possono essere in maiuscolo o minuscolo, ma questo dipende da DD e non da th:

- DDth o DDTH produce 11TH
- Ddth o DdTH produce 11Th
- ddth o ddTH produce 11th

Questo vale per tutti i numeri nel 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 doppi apici vengono semplicemente inserite dove si trovano. Gli spazi tra queste richieste di formato sono riprodotti nel risultato (si considerino i tre spazi nell'esempio precedente prima della parola "in". YyyY è stato inserito per dimostrare che non c'è distinzione tra maiuscole e minuscole a meno che non venga utilizzato un suffisso come Th (ORACLE considera yyyy, Yyyy, yyyY, yYYy e yYYY come equivalenti).

Per semplicità, si consideri questa richiesta di formattazione:

```
select DataNascita, TO_CHAR(DataNascita,'Month, ddth, YyyY')
      Formattata
   from COMPLEANNO;
DATANASCITA FORMATTATA
-----
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 allineati, per cui è facile confrontare le righe. Questo è l'allineamento di default e ORACLE lo realizza riempiendo i nomi dei mesi con spazi fino alla larghezza di nove. Vi sono casi dove è più importante che una data sia formattata in un modo leggibile, ad esempio quando è inserita in alto in una lettera. Nell'esempio, gli spazi tra mese e virgola sembrerebbero strani. Per eliminarli si utilizza fm come prefisso alle parole month o day:

- Month, ddth produce August , 20th
- fmMonth, ddth produce August, 20th

- Day, ddth produce Monday , 20th
- fmDay, ddth produce Monday, 20th

Si consideri il seguente esempio:

```
select Datanascita, TO_CHAR(Datanascita,'fmMonth, ddth, YyyY')
Formattata
  from COMPLEANNO;
```

DATANASCITA FORMATTATA

```
-----  
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" dd fmMonth, YYYY, "alle" HH.MI "del mattino"'
  Formattata
  from COMPLEANNO
where Nome = 'VICTORIA';
```

| NOME | DATANASCITA FORMATTATA |
|----------|--|
| VICTORIA | 20-MAY-49 Bambina nata il 20 May, 1949, alle 3.27 del mattino |

Ora si supponga di voler stampare la data in lettere. Per fare questo occorre utilizzare il controllo sp:

```
select Nome, Datanascita, TO_CHAR(Datanascita,
'"Bambina nata il" Ddsptn "di" fmMonth, YYYY, "alle" HH.MI')
  Formattata
  from COMPLEANNO
where Nome = 'VICTORIA';
```

| NOME | DATANASCITA FORMATTATA |
|----------|--|
| 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 doppi apici, ma il risultato riporterebbe sempre “del mattino” o “del pomeriggio”, indipendentemente dall’ora effettiva di nascita (dal momento che i doppi apici racchiudono un letterale). Invece, ORACLE consente di aggiungere dopo l’ora “A.M.” o “P.M.”, ma non tra doppi apici, interpretandolo come una richiesta di visualizzare questi codici.

Nell'esempio seguente si può notare come questo controllo di formattazione è 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" Ddsptf "di" fmMonth, YYYY, "alle" HH.MI P.M.')
  Formattata from COMPLEANNO
where Nome = 'VICTORIA';
```

| NOME | DATANASCITA FORMATTATA |
|----------|---|
| VICTORIA | 20-MAY-49 Bambina nata il Twentieth di May, 1949, alle 3.27 A.M. |

Un elenco di tutte le opzioni possibili è riportato nel paragrafo “Formati delle date”. Come si fa a creare un formato di data per l'anno 769 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 poi basta semplicemente formattare il risultato utilizzando TO_CHAR.

Formati delle date

I seguenti formati di data sono utilizzati sia con TO_CHAR sia con TO_DATE.

| | |
|-------|--|
| MM | Numero del mese: 12 |
| RM | Numero romano del mese: XXII |
| MON | Abbreviazione di tre lettere per il mese: AUG |
| MONTH | Nome completo del mese: AUGUST |
| DDD | Numero del giorno nell'anno, dal 1 gennaio: 354 |
| DD | Numero del giorno nel mese: 23 |
| D | Numero del giorno nella settimana: 6 |
| DY | Abbreviazione di tre lettere per il giorno: FRY |
| DAY | Nome del giorno completo: FRIDAY |
| YYYY | Anno completo di quattro cifre: 1946 |
| SYYYY | Anno con segno, 1000 A.C.= -1000 |
| YY | Ultime tre cifre dell'anno: 946 |
| YY | Ultime due cifre dell'anno: 46 |
| Y | Ultima cifra dell'anno |
| IYYY | Anno di quattro cifre secondo lo standard ISO* |
| IYY | Anno di tre cifre secondo lo standard ISO |
| IY | Anno di due cifre secondo lo standard ISO |
| I | Anno di una cifra secondo lo standard ISO |
| RR | Ultime due cifre dell'anno relative alla data corrente |
| YEAR | L'anno scritto in lettere: NINETEEN-FORTY-SIX |

| | |
|---------|--|
| Q | Numero del trimestre: 3 |
| WW | Numero delle settimane nell'anno: 46 |
| IW | Settimane nell'anno secondo lo standard ISO |
| W | Numero delle settimane nel mese: 3 |
| J | Giuliano, giorni dal 31 dicembre 4713 A.C.: 2422220 |
| HH | Ore del giorno, sempre 1-12: 11 |
| HH12 | Lo stesso di HH |
| HH24 | Ore del giorno, su 24 ore |
| MI | Minuti di ora: 58 |
| SS | Secondi di minuto: 43 |
| SSSSS | Secondi dalla mezzanotte, sempre 0-86399: 43000 |
| /,-: | Segni di interpunkzione da incorporare nella visualizzazione per TO_CHAR o da ignorare nel formato per TO_DATE |
| A.M. | Mostra A.M. o P.M. a seconda dell'ora del giorno |
| P.M. | Lo stesso effetto di A.M. |
| AM o PM | Lo stesso di A.M. ma senza punti |
| B.C. | Mostra B.C. o A.C. a seconda della data |
| A.D. | Lo stesso di B.C. |
| BC o AD | Lo stesso di B.C. ma senza punti |

*ISO è l'International Standards Organization, che utilizza una serie differente di standard per le date rispetto ai formati degli Stati Uniti.

I seguenti formati di data funzionano solo con TO_CHAR e non con TO_DATE:

| | |
|----------|---|
| “string” | Stringa che viene incorporata nella visualizzazione con TO_CHAR |
| fm | Prefisso per Month o Day: fmMONTH o fmday. Elimina il riempimento per il mese o il giorno (definiti in precedenza) nel formato. Senza fm, tutti i mesi sono visualizzati con la stessa lunghezza e lo stesso vale per i giorni. Con fm, il riempimento è eliminato; la lunghezza dei mesi e dei giorni dipende dal numero di caratteri. |
| TH | Suffisso a un numero: ddTH o DDTH produce 24th o 24TH. Le maiuscole dipendono da DD e non da TH. Funziona con qualsiasi numero in una data: YYYY, DD, MM, HH, MI, SS e così via. |
| SP | Suffisso a un numero che induce a scrivere il numero in cifre: DDSP, DdSP o ddSP produce THREE, Three o three. le maiuscole dipendono da DD e non da SP. Funziona con qualsiasi numero in una data: YYYY, DD, MM, HH, MI, SS, ecc. |
| SPTH | Suffisso combinazione di TH e SP che induce un numero a essere rappresentato in cifre come ordinale: Ddspth produce Third. Le maiuscole dipendono da DD e non da SP. Funziona con qualsiasi numero in una data: YYYY, DD, MM, HH, MI, SS e così via. |
| THSP | Lo stesso di SPTH. |



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.

Ad 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. Però gli utenti spesso selezionano MM, forse perché stanno impiegando anche due altre coppie di lettere doppie, HH e SS. Se si inserisce MM, viene restituito il mese, non i minuti:

```
select TO_CHAR(SysDate,'HH.MM.SS') OraNo from DUAL;
```

```
OraNo
-----
10.04.30
```

Adesso l'ora non è corretta, dal momento che è stato selezionato il mese al posto dei minuti. 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 comunica che l'ora e la data di una colonna di date o le date letterali indicate sono di altri fusi orari. Attualmente questa funzione è corretta solo per i fusi orari compresi fra Greenwich, in Inghilterra, e le Hawaii.

Questa è la sintassi per NEW_TIME:

```
NEW_TIME(data,'questo','quello')
```

date è la data (e l'ora) nel fuso orario *questo*. *questo* deve essere sostituito da un'abbreviazione di tre lettere per il fuso orario corrente. *quello* deve 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? Dal momento che ogni data in ORACLE contiene anche l'ora, è sufficiente utilizzare TO_CHAR e NEW_TIME per mostrare le differenze 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') Nascita  
   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 20 MAY, 1949, in un formato di data di ORACLE, consentendo così, di utilizzarla in calcoli di date.

Ecco la sintassi di TO_DATE:

```
TO_DATE(stringa[, 'formato'])
```

Per convertire la stringa 22-FEB-98 in un formato di data di ORACLE, si utilizza quanto segue:

```
select TO_DATE('22-FEB-98', 'DD-MON-YY') from DUAL;
```

```
TO_DATE(  
-----  
22-FEB-98
```

Occorre notare, però, che il formato 22-FEB-98 è già quello di default 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-98') from DUAL;
```

```
TO_DATE(  
-----  
22-FEB-98
```

Ma qual è il secolo della data? Se non si specifica il valore completo di quattro cifre per l'anno, allora il valore appropriato del secolo dipende dalle impostazioni di default del database.

Se la stringa è in un formato simile, ma non proprio quello di default DD-MON-YY, TO_DATE fallisce:

```
select TO_DATE('02/22/98') from DUAL;
```

```
ERROR: ORA-1843:  not a valid month
```

Quando il formato (*formato*) corrisponde alla stringa letterale (*stringa*), questa viene convertita in data e visualizzata nel formato di default della data:

```
select TO_DATE('02/22/98','MM/DD/YY') from DUAL;
```

```
TO_DATE('
```

```
-----
```

```
22-FEB-98
```

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-98','Day') from DUAL
```

```
*
```

```
ERROR at line 1: ORA-1841:  invalid TO_CHAR format string
```

Il messaggio è un po' fuorviante, ma il punto è che la query fallisce. Perché 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-98'),'Day') from DUAL
```

```
TO_CHAR(TO_DATE('22-FEB-98'),'DAY')
```

```
-----
```

```
Sunday
```

TO_DATE può accettare anche numeri, senza gli apici singoli, invece di stringhe, purché siano formattati in modo coerente. Ecco un esempio:

```
select TO_DATE(11051946,'MM DD YYYY') from DUAL;
```

```
TO_DATE(1
```

```
-----
```

```
05-NOV-46
```

I segni di interruzione nel formato vengono ignorati, ma il numero deve seguire l'ordine dei controlli di formato, senza segni di interruzione.

Quanto può essere complesso il controllo di formato in TO_DATE? Si supponga di voler semplicemente invertire l'istruzione select con TO_CHAR mostrata prima, mettendo il risultato in una parte *stringa* di TO_DATE e conservando lo stesso formato di TO_CHAR.

```
select TO_DATE('Bambina nata il Twentieth di May, 1949, alle 3.27 A.M.',  
'"Bambina nata il" Ddsph "di" fmMonth, YYYY, "alle" HH.MI P.M.')  
    Formattata
```

```
from COMPLEANNO  
where Nome = 'VICTORIA';
```

ERROR: ORA-1858: a letter was found in a date where a number was expected

Si ha un errore perché, quando la data viene riconvertita, può essere utilizzato solo un numero limitato di controlli di formato.

Infatti vi sono delle restrizioni sul formato che controlla TO_DATE:

- Non sono consentite stringhe letterali, come “Bambina nata il”.
- I giorni non possono essere scritti a lettere. Devono essere dei numeri.
- Sono consentiti i segni di interpunzione.
- fm non è necessario. Se utilizzato, viene ignorato.
- Se viene utilizzato Month, il mese nella stringa deve essere scritto in lettere. Se viene utilizzato Mon, il mese deve essere un’abbreviazione di tre lettere. Le lettere maiuscole e minuscole vengono ignorate.

Questa istruzione select funziona:

```
select TO_DATE('August, 20, 1949, 3.27 A.M. ', 'fmMonth, Dd,  
YYYY, HH.MI P.M. ') Formattata  
from COMPLEANNO  
where Nome = 'VICTORIA';
```

FORMATATA

20-AUG-49

8.4 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, Dataeff, DataCelebrata  
from Festa  
where DataCelebrata - Dataeff != 0;
```

| FESTA | DATAEFF | DATACELEBRATA |
|-------------------------|-----------|---------------|
| MARTIN LUTHER KING, JR. | 15-JAN-95 | 16-JAN-95 |
| LINCOLNS BIRTHDAY | 12-FEB-95 | 20-FEB-95 |
| WASHINGTONS BIRTHDAY | 22-FEB-95 | 20-FEB-95 |
| MEMORIAL DAY | 30-MAY-95 | 29-MAY-95 |
| COLUMBUS DAY | 08-OCT-95 | 09-OCT-95 |

Le date possono essere impiegate anche con altri operatori logici di ORACLE, con 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
      TO_DATE('01-JAN-1995','DD-MON-YYYY') and
      TO_DATE('22-FEB-1995','DD-MON-YYYY');
```

| FESTA | DATACELEBRATA |
|-------------------------|---------------|
| NEW YEAR DAY | 01-JAN-95 |
| MARTIN LUTHER KING, JR. | 16-JAN-95 |
| LINCOLNS BIRTHDAY | 20-FEB-95 |
| WASHINGTONS BIRTHDAY | 20-FEB-95 |
| FAST DAY, NEW HAMPSHIRE | 22-FEB-95 |

L'operatore logico IN funziona anche con stringhe letterali:

```
select Festa, DataCelebrata
  from FESTA
 where DataCelebrata IN ('01-JAN-95', '22-FEB-95');
```

| FESTA | DATACELEBRATA |
|-------------------------|---------------|
| NEW YEAR DAY | 01-JAN-95 |
| FAST DAY, NEW HAMPSHIRE | 22-FEB-95 |

Se non si è sicuri che il 1900 sia il secolo di default, 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-1995','DD-MON-YYYY'),
       TO_DATE('22-FEB-1995','DD-MON-YYYY'));
```

| FESTA | DATACELEBRATA |
|-------------------------|---------------|
| NEW YEAR DAY | 01-JAN-95 |
| FAST DAY, NEW HAMPSHIRE | 22-FEB-95 |

LEAST e GREATEST non funzionano, perché presuppongono che le stringhe letterali siano stringhe, non date.

```
select Festa, DataCelebrata
  from FESTA
 where DataCelebrata = LEAST('16-JAN-95', '04-SEP-95');
```

| FESTA | DATACELEBRATA |
|-----------|---------------|
| LABOR DAY | 04-SEP-95 |

Per fare in modo che LEAST e GREATEST funzionino correttamente, occorre applicare alle stringhe letterali la funzione TO_DATE. Tutti gli altri operatori logici funzionano correttamente con date specificate nel formato di default di ORACLE per le date, DD-MON-YY.

8.5 L'anno 2000

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.

In ORACLE tutti i valori di date hanno il secolo. Se si specificano soltanto le ultime due cifre dell'anno, ORACLE per default utilizza il secolo corrente quando inserisce il record. Ad esempio, il seguente listato mostra un inserimento nella tabella COMPLEANNO. Durante l'inserimento non viene specificato alcun valore del secolo per la colonna Datanascita.

```
insert into COMPLEANNO
  (Nome, Cognome, Datanascita, Eta)
  values
    ('ALICIA', 'ANN', '21-NOV-39', null);
```

Nell'esempio precedente non è specificato alcun valore per il secolo nella colonna Datanascita e nessuna età. Se si utilizza la funzione TO_CHAR nella colonna Datanascita, si può vedere la data di nascita completa inserita da ORACLE, con il secolo impostato al valore corrente per default.

```
select TO_CHAR(Datanascita,'DD-MON-YYYY') Nata
  from COMPLEANNO
 where Nome='ALICIA'
   and Cognome='ANN';
```

```
NATA
-----
21-NOV-1939
```

Per le date che possono essere correttamente impostate per default al secolo corrente, l'utilizzo del valore di default non presenta problemi. Ma che cosa succede nell'anno 2000? In tale anno il valore di default per il secolo corrente è 20, non 19. L'inserimento precedente, effettuato nell'anno 2000, produrrebbe come valore di Datanascita 21-NOV-2039, 39 anni avanti nel futuro.

Non occorre attendere fino all'anno 2000 per iniziare a sperimentare problemi con i secoli di default. Se l'applicazione registra le date di eventi futuri (come anniversari o date di scadenza di prodotti), per qualsiasi valore di anno che va oltre il 2000 possono verificarsi dei problemi. Ad esempio, se si cerca di salvare come data di scadenza di un prodotto il giorno 01-FEB-01 e l'anno corrente del sistema è il 1998, ORACLE memorizza l'anno come 1901, ovvero il prodotto risulta scaduto 97 anni fa.

Ogni volta che si utilizzano le date, si dovrebbero specificare i valori dell'anno con quattro cifre. Se questo non è possibile, si dovrebbe controllare il metodo che ORACLE impiega per assegnare i valori di default del secolo per la data. Il formato RR può essere d'aiuto, poiché arrotonda le due cifre dell'anno al secolo più vicino. Perciò, i valori dell'anno da 50 a 99 utilizzano il secolo corrente ('99' diventa '1999') mentre quelli da 00 a 49 sono arrotondati al secolo successivo ('01' diventa '2001').

Il seguente listato mostra la formattazione della data '02-JAN-00' utilizzando sia YY sia RR.

```
select TO_CHAR(TO_DATE('000102','RRMMDD'),'DD-Mon-YYYY') RRMMDD,  
       TO_CHAR(TO_DATE('000102','YYMMDD'),'DD-Mon-YYYY') YYMMDD  
  from DUAL;
```

| RRMMDD | YYMMDD |
|----------------------|-------------|
| ----- 02-Jan-2000 | 02-Jan-1900 |

Il formato RR può essere utilizzato con TO_CHAR e TO_DATE e può anche essere specificato a livello di istanza attraverso il parametro NLS_DATE_FORMAT nel file INIT.ORA. Per evitare problemi relativi al secolo, si dovrebbe sempre specificare quest'ultimo quando si inserisce una data. Se è necessario utilizzare il formato RR, è opportuno farlo in casi isolati in cui non è possibile impiegare anni di quattro cifre.

• Capitolo 9

• **Funzioni di conversione e trasformazione**

- 9.1 **Funzioni elementari di conversione**
• 9.2 **Funzioni di conversione specializzate**
• 9.3 **Funzioni di trasformazione**
• 9.4 **Riepilogo**

In questo capitolo vengono esaminate le funzioni che convertono un tipo di dati in un altro.

In precedenza sono stati trattati esaurientemente quattro tipi di dati e le loro funzioni associate: CHAR (stringhe di caratteri di lunghezza fissa), VARCHAR2 (stringhe di caratteri di lunghezza variabile), NUMBER e DATE.

- CHAR e VARCHAR2 includono qualsiasi lettera dell'alfabeto, qualsiasi numero o qualsiasi simbolo sulla tastiera. I letterali carattere devono essere racchiusi tra apici singoli: '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 di default DD-MON-YY, ma può essere visualizzato in molti modi utilizzando la funzione TO_CHAR, come si è mostrato nel Capitolo 8. I letterali DATE devono essere racchiusi fra apici singoli: '26-AUG-81'.

Ognuno di questi tipi di dati ha un gruppo di funzioni progettate appositamente per la loro manipolazione, elencate nella Tabella 9.1. Le funzioni stringa sono utilizzate con colonne o letterali di caratteri e 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, cioè modificano gli oggetti. Però quelle trattate in questo capitolo (Tabella 9.2) cambiano gli oggetti in modo inusuale: li trasformano da un tipo di dati a un altro o effettuano una profonda trasformazione dei dati.

L'utilizzo di due di queste funzioni, TO_CHAR e TO_DATE è stato già esaminato nel Capitolo 8. TO_CHAR trasforma una data in una stringa di caratteri (nel formato richiesto) e può convertire non solo date, ma anche numeri in stringhe di caratteri. Anche TO_DATE è una funzione di trasformazione, infatti converte una stringa di caratteri o un numero nel tipo di dati DATE. Può anche essere utilizzata nell'aritmetica delle date per calcolare MONTHS_BETWEEN, NEXT_DAY e altre funzioni sulle date.

Tabella 9.1 Funzioni suddivise per tipi di dati. (*continua*)

| FUNZIONI DI STRINGA PER TIPI DI DATI CHAR E VARCHAR2 | FUNZIONI ARITMETICHE PER IL TIPO DI DATI NUMBER | FUNZIONI DI DATA PER IL TIPO DI DATI DATE |
|---|--|--|
| (concatenamento) | + (addizione) | ADD_MONTHS |
| ASCII | - (sottrazione) | LAST_DAY |
| CHR | * (moltiplicazione) | MONTHS_BETWEEN |
| CONCAT | / (divisione) | NEW_TIME |
| CONVERT | ABS | NEXT_DAY |
| INITCAP | CEIL | ROUND |
| INSTR | COS | TRUNC |
| INSTRB | COSH | |
| LENGTH | EXP | |
| LENGHTB | FLOOR | |
| LOWER | LN | |
| LPAD | LOG | |
| LTRIM | MOD | |
| NLS_INITCAP | POWER | |
| NLS_LOWER | ROUND | |
| NLS_UPPER | SIGN | |
| NLSORT | SIN | |
| REPLACE | SINH | |
| RPAD | SQRT | |
| RTRIM | TAN | |
| SOUNDEX | TANH | |
| SUBSTR | TRUNC | |
| SUBSTRB | | |
| TRANSLATE | | |
| UID | | |
| UPPER | | |
| USER | | |
| USERENV | | |
| FUNZIONI DI GRUPPO | FUNZIONI DI CONVERSIONE | FUNZIONI VARIE |
| AVG | CHARTOROWID | DECODE |
| COUNT | CONVERT | DUMP |

Tabella 9.1 Funzioni suddivise per tipi di dati.

| FUNZIONI DI GRUPPO | FUNZIONI DI CONVERSIONE | FUNZIONI VARIE |
|--------------------|-------------------------|----------------|
| GLB | HEXTORAW | GREATEST |
| LUB | RAWTOHEX | GREATEST_LB |
| MAX | ROWIDTOCHAR | LEAST |
| MIN | TO_CHAR | LEAST_UB |
| STDDEV | TO_DATE | NVL |
| SUM | TO_MULTI_BYTE | VSIZE |
| VARIANCE | TO_NUMBER | |
| | TO_SINGLE_BYTE | |

Tabella 9.2 Funzioni di conversione e trasformazione.

| NOME DELLA FUNZIONE | DEFINIZIONE |
|---------------------|---|
| CHARTOROWID | Converte una stringa di caratteri in modo che si comporti come un identificativo di riga interno di ORACLE, o Rowld. |
| CONVERT | Converte una stringa di caratteri da una lingua a un'altra. |
| DECODE | Decodifica un CHAR, VARCHAR2 o NUMBER in varie differenti stringhe di caratteri o numeri, in base al valore. Questa è una funzione IF, THEN, ELSE molto potente, a cui è dedicato il Capitolo 16. |
| HEXTORAW | Converte una stringa di caratteri di numeri esadecimali in binario. |
| RAWTOHEX | Converte una stringa di numeri binari in una stringa di caratteri di numeri esadecimali. |
| ROWIDTOCHAR | Converte un identificativo di riga interno di ORACLE, o Rowld, in modo che si comporti come una stringa di caratteri. |
| TO_CHAR | Converte un NUMBER o DATE in modo che si comporti come una stringa di caratteri. |
| TO_DATE | Converte un NUMBER, CHAR o VARCHAR2 in modo che si comporti come un DATE (un tipo di dati particolare di ORACLE). |
| TO_MULTI_BYTE | Converte caratteri di un singolo byte in caratteri di più byte. |
| TO_NUMBER | Converte un CHAR o VARCHAR2 in modo che si comporti come un numero. |
| TO_SINGLE_BYTE | Converte caratteri di più byte in caratteri di un singolo byte. |
| TRANSLATE | Traduce i caratteri di una stringa in caratteri diversi. |

9.1 Funzioni elementari di conversione

Esistono tre funzioni elementari di ORACLE per convertire un tipo di dati in un altro, descritte di seguito.

- TO_CHAR trasforma un DATE o un NUMBER in una stringa di caratteri;
- TO_DATE trasforma un NUMBER, un CHAR o un VARCHAR2 in un DATE;
- TO_NUMBER trasforma un CHAR o un VARCHAR2 in un NUMBER.

Perché queste trasformazioni sono importanti? TO_DATE ovviamente è necessaria per 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 impiegare 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 salvato 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
```

Qui la funzione TO_CHAR trasforma il numero puro 948033515 (si noti che non ha apici singoli intorno, come dovrebbe avere una stringa CHAR o VARCHAR2) in una stringa di caratteri. Poi SUBSTR taglia le posizioni da 1 a 5 di questa “stringa”, producendo 94803. Un trattino viene concatenato all'estremità destra e un altro 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. Si tratta di un codice di avviamento postale nel formato americano.

La funzione TO_CHAR consente di utilizzare le funzioni di manipolazione delle stringhe su numeri (e date) come se fossero effettivamente stringhe. Tuttavia, si consideri quanto segue:

```
select SUBSTR(948033515,1,5)|| '-' ||
       SUBSTR(948033515,6) Cap
  from DUAL;
```

```
CAP
-----
94803-3515
```

Ci si aspetterebbe un errore, perché 948033515 è un numero e non una stringa di caratteri. Però la funzione SUBSTR funziona comunque. E funzionerebbe con una reale colonna di database di tipo NUMBER? Ecco una tabella con una colonna Cap definita come NUMBER:

```
describe INDIRIZZO
```

| Name | Null? | Type |
|----------|-------|--------------|
| COGNOME | | VARCHAR2(25) |
| NOME | | VARCHAR2(25) |
| VIA | | VARCHAR2(50) |
| CITTA | | VARCHAR2(25) |
| PROV | | CHAR(2) |
| CAP | | NUMBER |
| TELEFONO | | VARCHAR2(12) |
| PRO | | VARCHAR2(5) |

Ora viene selezionato 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 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; è anche allineato a destra, come avviene per default per i numeri. Ma la colonna RTRIM è allineata a sinistra, proprio come le stringhe, e sono stati rimossi gli zero e i due dal lato destro dei codici di avviamento postale. Vi è qualcos'altro di particolare, in questo caso. La sintassi di RTRIM, richiamata dal Capitolo 6, è la seguente:

RTRIM(stringa [, 'set'])

La serie (*set*) di caratteri da rimuovere dalla stringa è racchiusa fra apici singoli, ma in questo esempio:

RTRIM(Cap,20)

non vi sono apici. Come mai funziona?

Conversione automatica dei tipi di dati

ORACLE converte automaticamente i numeri riportati in precedenza, Cap e 20, in stringhe, come se fossero preceduti dalla funzione TO_CHAR. In effetti ORACLE trasforma automaticamente, con alcune evidenti eccezioni, qualsiasi tipo di dati in un altro, in base alla funzione che gli viene applicata. Se si tratta di una funzione stringa, ORACLE converte un NUMBER o un DATE immediatamente in una stringa e la funzione opera correttamente. Se si ha una funzione di data e la colonna o il letterale è una stringa nel formato DD-MON-YY, ORACLE la converte in una data. 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.

Ma il meccanismo non funziona sempre. Per ottenere una conversione automatica di un tipo di dati in un altro, il primo tipo di dati deve “assomigliare” a quello in cui deve essere convertito.

Le linee guida fondamentali sono fornite nel paragrafo “Direttive per la conversione automatica di tipi di dati”. Potrebbero essere un po’ confuse, perciò vengono presentati alcuni esempi che aiutano a chiarirle. 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))  
-----  
01-Apr-98
```

Occorre notare che la funzione INITCAP rende maiuscola la prima lettera di “apr” anche se si trova in mezzo alla stringa “01-apr-98”. 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 e 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 stringa:

```
select SUBSTR(SysDate,4,3) from DUAL;  
  
SUB  
---  
APR
```

Qui una data è riempita a sinistra con 9 in modo da ottenere una lunghezza totale di 20:

```
select LPAD(SysDate,20,'9') from DUAL;
```

```
LPAD(SYSDATE,20,'9')
-----
```

```
999999999901-APR-98
```

Si sarebbe ottenuto lo stesso risultato anche se il carattere da utilizzare per il riempimento, 9, non fosse stato racchiuso tra apici, perché 9 è un letterale NUMBER. Se si desidera effettuare il riempimento con caratteri, gli apici sono necessari:

```
select LPAD(SysDate,20,'-') from DUAL;
```

```
LPAD(SYSDATE,20,'9')
-----
```

```
-----01-APR-98
```

LPAD, o qualsiasi altra funzione stringa, può anche essere impiegata su NUMBER, letterali o colonne:

```
select LPAD(9,20,0) from DUAL;
```

```
LPAD(9,20,0)
-----
```

```
000000000000000000000009
```

Questi esempi mostrano che le funzioni stringa trattano NUMBER e DATE come se fossero stringhe di caratteri. Il risultato di una funzione (quello che viene visualizzato) è una stringa di caratteri. Nel prossimo esempio una stringa (si notino gli apici singoli) è 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. Funziona solo perché le stringhe letterali sono espresse nel formato di default delle date DD-MON-YY:

```
select MONTHS_BETWEEN('16-MAY-98','01-APR-98') from DUAL;
```

```
MONTHS_BETWEEN('16-MAY-98','01-APR-98')
-----
```

```
1.48387097
```

Una delle direttive riportate nel paragrafo “Direttive per la conversione automatica dei tipi di dati” afferma che una data non viene convertita in un NUMBER. Ma di seguito vi è un esempio di addizione e sottrazione con una data. Viola la direttiva?

```
select SysDate, SysDate+1, SysDate-1 from DUAL;
```

```
SYSDATE   SYSDATE+1  SYSDATE-1
----- ----- -----
```

```
01-APR-98 02-APR-98 31-MAR-98
```

La risposta è no, perché l'addizione e la sottrazione sono aritmetica delle date, non aritmetica normale. L'aritmetica delle date (trattata nel Capitolo 8) funziona solo con l'addizione e la sottrazione e solo con DATE.

La maggior parte delle funzioni converte automaticamente una stringa di caratteri nel formato di default delle date. Un'eccezione è questo tentativo di sommare una data con un letterale:

```
select '01-APR-98'+1 from DUAL;
```

```
ERROR: ORA-1722: 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-0932: inconsistent data types
```

Infine, un NUMBER non viene mai automaticamente convertito in una DATE, perché un numero puro non può essere espresso nel formato di default per un data, che è DD-MON-YY:

```
select NEXT_DAY(040198,'FRIDAY') from DUAL;  
*
```

```
ERROR at line 1: ORA-0932: inconsistent data types
```

Per utilizzare un NUMBER in una funzione di data, occorre TO_DATE.

Direttive per la conversione automatica di tipi di dati

Le seguenti direttive descrivono la conversione automatica di dati da un tipo all'altro, in base alla funzione in cui i dati sono utilizzati.

- Ogni NUMBER o DATE può essere convertito in una stringa di caratteri. Perciò, qualsiasi funzione stringa può essere utilizzata su una colonna NUMBER o DATE. I letterali NUMBER non devono essere racchiusi fra apici singoli, quando vengono utilizzati in una funzione di stringhe, i letterali DATE sì.
 - Un dato CHAR o VARCHAR2 viene convertito in DATE solo se è espresso nel formato DD-MON-YY, come 07-AUG-99. Questo vale per tutte le funzioni fatta eccezione per GREATEST e LEAST, che considerano il dato come una stringa; per BETWEEN vale solamente se la colonna a sinistra dopo la parola BETWEEN è una DATE. Altrimenti deve essere utilizzata la funzione TO_DATE, con un formato appropriato.
 - Una DATE non viene convertita in un NUMBER.
 - Un NUMBER non viene convertito in una DATE.
-

Un'avvertenza sulla conversione automatica

Il problema di consentire a SQL di effettuare la conversione automatica dei tipi di dati è controverso. Da una parte semplifica e riduce considerevolmente le funzioni necessarie per far funzionare l'istruzione select; dall'altra, se l'ipotesi sul contenuto della colonna è sbagliata (ad esempio, si suppone che una particolare colonna di caratteri contenga sempre un numero e quindi possa essere utilizzata nei calcoli), il report potrebbe a un certo punto interrompersi, producendo un messaggio di errore, e occorre perdere del 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.

Una semplice regola pratica potrebbe essere quella di utilizzare funzioni in cui il rischio è basso, 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.

9.2 Funzioni di conversione specializzate

In ORACLE vi sono varie funzioni di conversione specializzate. I loro nomi e formati sono i seguenti.

| | |
|--------------------------------|---|
| CHARTOROWID(<i>stringa</i>) | Converte una stringa di caratteri in un identificativo di riga. |
| ROWIDTOCHAR(<i>rowid</i>) | Converte un identificativo di riga in una stringa di caratteri. |
| HEXTORAW(<i>esadecimale</i>) | Converte un numero esadecimale in binario. |
| RAWTOHEX(<i>raw</i>) | Converte un numero binario in esadecimale. |
| TO_MULTI_BYTE | Converte caratteri di un singolo byte in caratteri di più byte. |
| TO_SINGLE_BYTE | Converte caratteri di più byte in caratteri di un singolo byte. |

Se si prevede di utilizzare SQLPLUS e ORACLE semplicemente per produrre report, queste funzioni non saranno mai necessarie. Se invece si impiega SQLPLUS per aggiornare il database o si intende costruire applicazioni ORACLE, oppure si utilizza National Language Support, queste informazioni probabilmente saranno importanti. Le funzioni possono essere trovate, per nome, nella guida alfabetica di riferimento riportata nella Parte 5 di questo libro.

9.3 Funzioni di trasformazione

Anche se in un certo senso ogni funzione che cambia il suo oggetto può essere considerata una funzione di trasformazione, vi sono due funzioni inusuali che possono

essere utilizzate in molti modi interessanti per controllare l'output in base all'input, invece di limitarsi a trasformarlo: si tratta di TRANSLATE e DECODE.

TRANSLATE

TRANSLATE è una semplice funzione che effettua una sostituzione ordinata carattere per carattere.

Ecco la sintassi di TRANSLATE:

```
TRANSLATE(stringa,if,then)
```

TRANSLATE esamina ogni carattere in *stringa*, poi 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 *string* con quello in *then*. Normalmente la funzione è scritta su una sola riga:

```
select TRANSLATE(7671234,234567890,'BCDEFGHIJ')  
      from DUAL;
```

```
TRANSLATE(7671234,234567890,'BCDEFGHIJ'  
-----  
GFG1BCD
```

Ma può essere più facile da capire se viene suddivisa in due riga (per SQLPLUS non fa differenza, naturalmente):

```
select TRANSLATE(7671234,234567890,  
                  'BCDEFGHIJ')  
      from DUAL;  
  
TRANSLATE(7671234,234567890,'BCDEFGHIJ'  
-----  
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 (è quello che succede con "1").

TRANSLATE è tecnicamente una funzione stringa, ma 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 tali metodi di sostituzione di caratteri per codificare i messaggi prima di inviarli. Il destinatario effettuava 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','ABCDEFGHIJKLMNPQRSTUVWXYZ',  
                  '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*. È una funzione incredibilmente potente, con un'ampia gamma di campi in cui può essere utile; il Capitolo 16 è interamente dedicato alla trattazione del suo utilizzo avanzato.

Ecco la sintassi di DECODE:

```
DECODE(valore,if1,then1,if2,then2,if3,then3,...,else)
```

Qui sono illustrate solo due combinazioni *if/then*, ma in realtà non vi è alcun limite per esse. Per vedere come opera questa funzione, viene qui richiamata la tabella GIORNALE citata in precedenza:

```
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 |
| Salute | F | 6 |
| Vita moderna | B | 1 |
| Fumetti | C | 4 |
| Film | B | 4 |
| Bridge | B | 2 |
| Necrologi | F | 6 |

Si supponga di voler cambiare il nome di una coppia di caratteristiche regolari. DECODE controlla ogni valore di Argomento, riga per riga. Se il valore che trova è ‘Sport’, lo sostituisce con ‘Giochi atletici’; se trova ‘Film’, lo sostituisce con ‘Spettacoli’, se trova qualsiasi altra cosa, utilizza il valore di Argomento.

Nell’esempio seguente viene decodificato il numero di pagina: se è 1, viene sostituito con la parola ‘Prima pagina’. Se è un qualsiasi altro valore, viene concatenato con la parola ‘Pagina’. Questo mostra tra l’altro 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','PRIMA 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 |

Vi sono alcune restrizioni per i tipi di dati di *if* e *then*, che verranno trattate nel Capitolo 16.

9.4 Riepilogo

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 attraverso una conversione automatica anche con altri tipi. 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 di NUMBER convertono un CHAR o un VARCHAR2, se contiene solo le cifre da 0 a 9, un punto decimale o un segno meno (-) a sinistra, mentre non convertono le DATE. Le funzioni di DATE convertono le stringhe di caratteri se sono nel formato DD-MON-YY, ma non convertono i NUMBER.

Due funzioni, TRANSLATE e DECODE, cambiano radicalmente i dati su cui agiscono. TRANSLATE effettua una sostituzione di caratteri in base a qualsiasi configurazione specificata e DECODE realizza una sostituzione di valori, sempre per qualsiasi configurazione specificata.

• Capitolo 10

• Raggruppamento di righe

• 10.1 Utilizzo di group by e having

• 10.2 Viste di gruppi

• 10.3 La potenza delle viste di gruppi

• 10.4 where, having, group by e order by

Finora si è visto come SQL possa 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. Si è visto anche come sia possibile convertire i valori nelle colonne tra tipi di dati carattere, NUMBER e DATE e come funzioni di gruppo possano fornire informazioni sull'intera serie di righe.

Oltre alle funzioni di gruppo, vi sono 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.

In questo capitolo, per rendere la visualizzazione delle informazioni coerente e leggibile, sono fornite a SQLPLUS le seguenti definizioni di colonna:

```
column Importo format 999.90
column Media format 999.90
column Articolo format a16
column Mese format a9
column Percento format 99.90
column Persona format a25
column Totale format 999.90
column TotaleAnno format 999.90
```

La Figura 10.1 presenta un elenco di date di pagamento (DataAzione), voci (Articolo), persone (Persona) e importi pagati (Importo) per ogni dipendente di G.B. Talbot durante il 1901. L'istruzione SQL che ha creato la Figura 10.1 è la seguente:

```
select Data, Articolo, Persona, Importo
  from REGISTRO
 where Azione='PAGATO'
 order by DataAzione;
```

Un attento esame di questa tabella rivela che molte delle voci sono ripetute.

| DATAAZION | ARTICOLO | PERSONA | IMPORTO |
|-----------|----------------|---------------------------|---------|
| 04-JAN-01 | LAVORARE | GERHARDT KENTGEN | 1.00 |
| 12-JAN-01 | LAVORARE | GEORGE OSCAR | 1.00 |
| 19-JAN-01 | LAVORARE | GERHARDT KENTGEN | 1.00 |
| 30-JAN-01 | LAVORARE | ELBERT TALBOT | .50 |
| 04-FEB-01 | LAVORARE | ELBERT TALBOT | .50 |
| 28-FEB-01 | LAVORARE | ELBERT TALBOT | 1.00 |
| 05-MAR-01 | LAVORARE | DICK JONES | 1.00 |
| 14-MAR-01 | ARARE | JED HOPKINS | 3.00 |
| 20-MAR-01 | LAVORARE | DICK JONES | 1.00 |
| 21-MAR-01 | LAVORARE | VICTORIA LYNN | 1.00 |
| 01-APR-01 | SEMINARE | RICHARD KOCH AND BROTHERS | 3.00 |
| 16-APR-01 | SEMINARE | RICHARD KOCH AND BROTHERS | 3.00 |
| 27-APR-01 | SEMINARE | RICHARD KOCH AND BROTHERS | 3.00 |
| 02-MAY-01 | LAVORARE | DICK JONES | 1.00 |
| 11-MAY-01 | LAVORARE | WILFRED LOWELL | 1.20 |
| 24-MAY-01 | LAVORARE | WILFRED LOWELL | 1.20 |
| 03-JUN-01 | LAVORARE | ELBERT TALBOT | 1.00 |
| 13-JUN-01 | LAVORARE | PETER LAWSON | 1.00 |
| 18-JUN-01 | TREBBIARE | WILLIAM SWING | .50 |
| 26-JUN-01 | DIPINGERE | KAY AND PALMER WALBOM | .25 |
| 14-JUL-01 | LAVORARE | WILFRED LOWELL | 1.20 |
| 15-JUL-01 | LAVORARE | KAY AND PALMER WALBOM | 2.25 |
| 28-JUL-01 | TAGLIARE | DICK JONES | .75 |
| 06-AUG-01 | SEMINARE | VICTORIA LYNN | 1.80 |
| 06-AUG-01 | SEMINARE | ANDREW DYE | 4.00 |
| 10-AUG-01 | LAVORARE | HELEN BRANDT | 1.00 |
| 11-AUG-01 | LAVORARE | HELEN BRANDT | 2.00 |
| 18-AUG-01 | SARCHIARE | ELBERT TALBOT | .90 |
| 22-AUG-01 | TAGLIARE | PETER LAWSON | 1.00 |
| 23-AUG-01 | TAGLIARE | PETER LAWSON | 1.00 |
| 09-SEP-01 | LAVORARE | ADAH TALBOT | 1.00 |
| 11-SEP-01 | LAVORARE | ROLAND BRANDT | .75 |
| 23-SEP-01 | ARARE | RICHARD KOCH AND BROTHERS | 1.50 |
| 29-SEP-01 | LAVORARE | GERHARDT KENTGEN | 1.00 |
| 07-OCT-01 | SEMINARE | RICHARD KOCH AND BROTHERS | 1.50 |
| 07-OCT-01 | LAVORARE | JED HOPKINS | 1.00 |
| 09-OCT-01 | LAVORARE | DONALD ROLLO | .63 |
| 22-OCT-01 | SEMINARE | DICK JONES | 1.80 |
| 07-NOV-01 | LEGNA TAGLIATA | ANDREW DYE | .50 |
| 10-NOV-01 | LAVORARE | JOHN PEARSON | .63 |
| 12-NOV-01 | LAVORARE | PAT LAVAY | 1.50 |
| 13-NOV-01 | TAGLIARE CEPPI | PAT LAVAY | .25 |
| 13-NOV-01 | ESTRARRE CEPPI | PAT LAVAY | .75 |
| 12-DEC-01 | LAVORARE | BART SARJEANT | 1.00 |
| 13-DEC-01 | LEGNA TAGLIATA | PAT LAVAY | .50 |
| 17-DEC-01 | TAGLIARE | DICK JONES | .75 |

46 rows selected

Figura 10.1 Date e importi pagati da G.B. Talbot nel 1901.

10.1 Utilizzo di group by e having

Se Talbot desidera un totale degli importi pagati, raggruppati per voci, deve scrivere una query come questa:

```
select Articolo, SUM(Importo) Totale, COUNT(Articolo)
  from REGISTRO
 where Azione = 'PAID'
 group by Articolo;
```

| ARTICOLO | TOTALE | COUNT(ARTICOLO) |
|----------------|--------|-----------------|
| TAGLIARE CEPPI | .25 | 1 |
| SCAVARE FOSSE | 3.00 | 1 |
| ARARE | 1.50 | 1 |
| ESTRARRE CEPPI | .75 | 1 |
| DIPINGERE | 1.75 | 1 |
| SEMINARE | 18.10 | 7 |
| LEGNA TAGLIATA | 1.00 | 2 |
| TAGLIARE | 3.50 | 4 |
| TREBBIARE | .50 | 1 |
| SARCHIARE | .90 | 1 |
| LAVORARE | 27.36 | 26 |

11 rows selected.

Si noti la combinazione del nome di una colonna, Articolo, e di due funzioni di gruppo, SUM e COUNT, nella clausola select. Questo è possibile solo perché ad Articolo si fa riferimento nella clausola group by. Se non fosse così, si otterrebbe l'oscuro messaggio incontrato per la prima volta nel Capitolo 7:

```
select Articolo, SUM(Importo) Totale, COUNT(Articolo)
*
ERROR at line 1: ORA-0937: not a single group set 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 Articolo nella clausola group by, che induce SUM a sommare tutte le righe raggruppate per ogni Articolo. COUNT comunica quante righe per ogni Articolo vi sono nel gruppo.

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 per cui la somma degli Importo, per gruppi di Articolo, è maggiore di 3 dollari.

```
select Articolo, SUM(Importo) Totale  from REGISTRO
  where Azione = 'PAID'  group by Articolo
 having SUM(Importo) > 3;
```

| ARTICOLO | TOTALE |
|-----------|--------|
| TREBBIARE | 18.10 |

| | |
|-----------|-------|
| SARCHIARE | 3.50 |
| LAVORARE | 27.36 |

Si possono anche trovare le voci per cui l'importo medio speso nel corso dell'anno è maggiore di quello medio per tutte le voci. Innanzitutto è opportuno controllare qual è la media (AVG) di tutte le voci:

```
select AVG(Importo) Media
  from REGISTRO
 where AZIONE='PAGATO';
MEDIA
-----
 1.27
```

Poi occorre incorporare questa media come una sottoquery (simile a quella effettuata nella clausola where nel Capitolo 3), per confrontare ogni gruppo di voci con questa media:

```
select Articolo, SUM(Importo) Totale, AVG(Importo) Media
  from REGISTRO
 where Azione = 'PAGATO'
 group by Articolo
 having AVG(Importo) > (select AVG(Importo)
                           from REGISTRO
                           where AZIONE = 'PAGATO');
```

| ARTICOLO | TOTALE | MEDIA |
|---------------|--------|-------|
| SCAVARE FOSSE | 3.00 | 3.00 |
| ARARE | 1.50 | 1.50 |
| DIPINGERE | 1.75 | 1.75 |
| SEMINARE | 18.10 | 2.59 |

Si noti che la clausola having confronta l'importo medio per ogni voce (perché le righe sono raggruppate per voci) con l'importo medio per tutte le voci pagate nel corso dell'anno. Si osservino anche le differenze e le somiglianze fra le colonne Totale e Media. Raggruppando i dati della tabella REGISTRO per voci si ottengono gli importi pagati e quelli medi per ogni voce. Dalla tabella REGISTRO di Talbot si possono ottenere anche informazioni simili per ogni persona:

```
select Persona, SUM(Importo) Totale
  from REGISTRO
 where Azione = 'PAGATO'
 group by Persona;
```

| PERSONA | TOTALE |
|---------------|--------|
| ADAH TALBOT | 1.00 |
| ANDREW DYE | 4.50 |
| BART SARJEANT | 1.00 |
| DICK JONES | 6.30 |
| DONALD ROLLO | .63 |
| ELBERT TALBOT | 3.90 |

| | |
|---------------------------|-------|
| GEORGE OSCAR | 1.00 |
| GERHARDT KENTGEN | 3.00 |
| HELEN BRANDT | 3.00 |
| JED HOPKINS | 4.00 |
| JOHN PEARSON | .63 |
| KAY AND PALMER WALBOM | 4.00 |
| PAT LAVAY | 3.00 |
| PETER LAWSON | 3.00 |
| RICHARD KOCH AND BROTHERS | 12.00 |
| ROLAND BRANDT | .75 |
| VICTORIA LYNN | 2.80 |
| WILFRED LOWELL | 3.60 |
| WILLIAM SWING | .50 |

19 rows selected.

Questo è un utile riepilogo in ordine alfabetico per persona, ma è possibile utilizzare un altro ordine di visualizzazione? Non si può utilizzare quello seguente:

group by SUM(Importo)

poiché le righe di ogni persona non sono più raggruppate insieme. Inoltre, lo scopo di group by non è produrre l'ordine desiderato, ma raggruppare elementi simili; perciò non può essere utilizzata per cambiare l'ordine.

Aggiunta di order by

La soluzione consiste nell'aggiungere una clausola order by dopo la clausola having. Si potrebbe aggiungere quella seguente:

order by Persona desc

che inverte l'ordine dell'elenco, oppure questa:

order by SUM(Importo) desc

Ad esempio, nel 1901 hanno lavorato per Talbot 19 persone, svolgendo 46 compiti nel corso dell'anno (Figura 10.1). A quali lavoratori Talbot ha pagato più di un dollaro, chi è stato pagato di più e qual è l'ordine relativo dei lavoratori in base alle entrate?

```
select Persona, SUM(Importo) Totale
  from REGISTRO
 where Azione = 'PAGATO'
 group by Persona
having SUM(Importo) > 1.00
 order by SUM(Importo) desc;
```

| PERSONA | TOTALE |
|---------------------------|--------|
| RICHARD KOCH AND BROTHERS | 12.00 |
| DICK JONES | 6.30 |
| ANDREW DYE | 4.50 |

| | |
|-----------------------|------|
| JED HOPKINS | 4.00 |
| KAY AND PALMER WALBOM | 4.00 |
| ELBERT TALBOT | 3.90 |
| WILFRED LOWELL | 3.60 |
| GERHARDT KENTGEN | 3.00 |
| PETER LAWSON | 3.00 |
| PAT LAVAY | 3.00 |
| HELEN BRANDT | 3.00 |
| VICTORIA LYNN | 2.80 |

12 rows selected.

Un metodo alternativo per comunicare a SQL quale colonna utilizzare per order by consiste nello specificarne il numero. La clausola order by della query precedente poteva essere scritta nel modo seguente:

order by 2 desc

Se SQL riscontra in order by un numero, esamina l'istruzione select e conta le colonne elencate, da sinistra a destra. La seconda colonna nell'esempio precedente era SUM(Importo), perciò SQL utilizza questa come colonna in base alla quale ordinare i risultati.

NOTA *A partire da ORACLE7.1, per un cambiamento nello standard ANSI SQL, la possibilità di utilizzare i numeri di posizione delle colonne nelle clausole order by verrà progressivamente eliminata. ORACLE7.1 e le versioni successive supportano l'utilizzo di alias di colonne (come Totale) nella clausola order by, che come metodo per specificare l'ordine è più affidabile del numero della posizione. Si consulti il paragrafo dedicato a order by nella guida alfabetica di riferimento.*

Ordine di esecuzione

La query precedente ha 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 colonne in base alla clausola where.
2. Queste colonne vengono raggruppate in base a 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 in order by. order by deve utilizzare una funzione di gruppo oppure una colonna specificata nella clausola group by.

Questo ordine di esecuzione è importante perché ha un impatto diretto sulle prestazioni delle query. In generale, più record possono essere eliminati attraverso le clausole where e più velocemente vengono eseguite le query. Questo guadagno di prestazioni è dovuto alla riduzione del numero di colonne che devono essere elaborate durante l'operazione di group by.

Se una query utilizza una clausola having per eliminare gruppi, conviene controllare se questa condizione può essere riscritta come una clausola where. In molti casi questa riscrittura non è possibile; di solito è realizzabile solo quando la clausola having è impiegata per eliminare gruppi basati sulle colonne di raggruppamento.

Ad esempio, se si avesse questa query:

```
select Persona, SUM(Importo) Totale
  from REGISTRO
 where Azione='PAGATO'
   group by Persona
 having Persona like 'P%'
   order by SUM(Acount) desc;
```

l'ordine di esecuzione sarebbe il seguente:

1. Eliminare le righe basate su:

```
where Azione='PAID'
```

2. Raggruppare le restanti righe in base a:

```
group by Persona
```

3. Per ogni Persona, calcolare:

```
SUM(Acount)
```

4. Eliminare i gruppi in base a:

```
having Persona like 'P%'
```

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 colonne (passaggio 2), devono essere svolti meno calcoli (passaggio 3) e non deve essere eliminato alcun gruppo (passaggio 4). Ognuno di questi passaggi viene eseguito più velocemente.

Dal momento che la condizione having in questo esempio non è basata su una colonna calcolata, può essere facilmente sostituita da una condizione where:

```
select Persona, SUM(Importo) Totale
  from REGISTRO
 where Azione='PAGATO'
   and Persona like 'P%'
   group by Persona
   order by SUM(Acount) desc;
```

10.2 Viste di gruppi

Nel Capitolo 3, all'inizio dell'esame di SQL e SQLPLUS, è stata creata una vista di nome INVASIONE per ORACLE , che univa le tabelle CLIMA e LOCAZIONE.

Questa vista è una tabella a tutti gli effetti, con colonne e righe, ma ognuna delle righe contiene colonne che in realtà provengono 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. Ad esempio, si consideri la seguente query di gruppi:

```
select LAST_DAY(DataAzione) MESE,
       SUM(Importo) Totale
  from REGISTRO
 where Azione = 'PAGATO'
 group by LAST_DAY(DataAzione);
```

| MESE | TOTALE |
|-----------|--------|
| 31-JAN-01 | 3.50 |
| 28-FEB-01 | 1.50 |
| 31-MAR-01 | 6.00 |
| 30-APR-01 | 9.00 |
| 31-MAY-01 | 3.40 |
| 30-JUN-01 | 4.25 |
| 31-JUL-01 | 4.20 |
| 31-AUG-01 | 11.70 |
| 30-SEP-01 | 4.25 |
| 31-OCT-01 | 4.93 |
| 30-NOV-01 | 3.63 |
| 31-DEC-01 | 2.25 |

12 rows selected.

La situazione è abbastanza semplice. Questa è una tabella (tecnicamente una vista) degli importi mensili pagati da Talbot per tutte le voci. La seguente istruzione:

LAST_DAY(AzioneDate)

fa in modo che ogni DataAzione reale sia trasformata nell'ultimo giorno del mese. Se DataAzione fosse stata la colonna in select, si sarebbero ottenuti i totali parziali giornalieri, invece che mensili. Lo stesso ordinamento si sarebbe potuto ottenere con:

TO_CHAR(DataAzione,'MON')

ma il risultato sarebbe stato fornito per mese in ordine alfabetico, e non secondo l'ordine normale dei mesi. Si può assegnare a questo risultato un nome e creare una vista:

```
create or replace view TOTALEMESE as
select LAST_DAY(DataAzione) MESE,
       SUM(Importo) Totale
  from REGISTRO
 where Azione = 'PAGATO'
 group by LAST_DAY(DataAzione);
```

View created.

Come rinominare le colonne con alias

Si notino i nomi MESE e Totale nella clausola select, che rinominano le colonne che li precedono. Questi nuovi nomi sono chiamati *alias*, perché sono utilizzati per nascondere i nomi reali delle colonne sottostanti (che sono complicati perché contengono funzioni). Il meccanismo è simile a quello descritto nel Capitolo 5 nel report SQLPLUS della Figura 5.3, dove Quantita per Tasso è stata rinominata Pro:

```
Quantita * Tasso Pro
```

In effetti, ora che questa vista è stata creata, può essere descritta in questo modo:

```
describe TOTALEMESE
```

| Name | Null? | Type |
|--------|-------|--------|
| MESE | | DATE |
| TOTALE | | NUMBER |

Quando si effettua una query sulla tabella, si possono (e si devono) utilizzare i nuovi nomi delle colonne:

```
select Mese, Totale
from TOTALEMESE;
```

| MESE | TOTALE |
|-----------|--------|
| 31-JAN-01 | 3.50 |
| 28-FEB-01 | 1.50 |
| 31-MAR-01 | 6.00 |
| 30-APR-01 | 9.00 |
| 31-MAY-01 | 3.40 |
| 30-JUN-01 | 4.25 |
| 31-JUL-01 | 4.20 |
| 31-AUG-01 | 11.70 |
| 30-SEP-01 | 4.25 |
| 31-OCT-01 | 4.93 |
| 30-NOV-01 | 3.63 |
| 31-DEC-01 | 2.25 |

12 rows selected.

“Totale” è uno alias di colonna e costituisce un altro nome da utilizzare per fare riferimento alla colonna. Da ORACLE7.1, gli alias delle colonne possono essere specificati in un modo leggermente diverso, cioè identificandoli nelle query attraverso la clausola as. Per fare questo si dovrebbe cambiare una parte del comando select nel precedente esempio:

```
SUM(Importo) AS Totale
```

L'utilizzo della clausola as aiuta a distinguere visivamente gli alias dalle colonne stesse.

Nella descrizione della vista e nella query non vi è alcuna traccia delle funzioni LAST_DAY(DataAzione) e SUM(Importo), ma sono presenti solo i loro nuovi nomi, MESE e TOTALE. È come se TOTALEMESE 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 la precede. Quindi trasforma la parola (l'alias) in maiuscolo, indipendentemente dal modo in cui è stato digitato. Si può eseguire una verifica confrontando i nomi delle colonne nei comandi create view e describe. Anche se in create view MESE è stato digitato in maiuscolo e Totale solo con l'iniziale maiuscola, nella descrizione della tabella che ORACLE conserva internamente sono entrambi in maiuscolo. Quando si crea un vista, non si devono mai racchiudere gli alias delle colonne fra doppi apici. In questo modo vengono salvati in maiuscolo, cosa necessaria perché ORACLE possa trovarli. Per un avvertimento sugli alias, si rimanda al paragrafo “Alias nella creazione di viste”.

Ora si hanno i totali mensili raccolti in una vista. Potrebbe anche essere creato un totale per l'intero anno, utilizzando TOTALEANNO sia come nome della vista che come alias della colonna SUM(Importo):

```
create or replace view TOTALEANNO as
select SUM(Importo) TOTALEANNO
  from REGISTRO
 where Azione = 'PAGATO';
```

View created.

Se si esegue una query nella vista, si scopre che ha un solo record:

```
select TOTALEANNO
  from TOTALEANNO;
```

```
TOTALEANNO
-----
58.61
```

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.

Perciò, quando si creano alias per una vista, non si devono mai utilizzare i doppi apici.

10.3 La potenza delle viste di gruppi

Ora si vedrà la reale potenza di un database relazionale. Sono state create delle viste del libro mastro di Talbot che contengono i totali per gruppi: per Articolo, per Persona, per Mese e per Anno. 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. Ad esempio, quale percentuale dei pagamenti dell'anno è stata effettuata in ogni mese?

```
select Mese, Totale, (Totale/TotaleAnno)*100 Percento
  from TOTALEMESE, TOTALEANNO
 order by Mese;
```

| MESE | TOTALE | PERCENTO |
|-----------|--------|----------|
| 31-JAN-01 | 3.50 | 5.97 |
| 28-FEB-01 | 1.50 | 2.56 |
| 31-MAR-01 | 6.00 | 10.24 |
| 30-APR-01 | 9.00 | 15.36 |
| 31-MAY-01 | 3.40 | 5.80 |
| 30-JUN-01 | 4.25 | 7.25 |
| 31-JUL-01 | 4.20 | 7.17 |
| 31-AUG-01 | 11.70 | 19.96 |
| 30-SEP-01 | 4.25 | 7.25 |
| 31-OCT-01 | 4.93 | 8.41 |
| 30-NOV-01 | 3.63 | 6.19 |
| 31-DEC-01 | 2.25 | 3.84 |

12 rows selected.

In questa query sono elencate due viste 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, `TOTALEANNO`, restituisce soltanto una riga (come mostrato nel listato precedente). Le viste `TOTALEMESE` e `TOTALEANNO` sono state create con l'ipotesi che nella tabella `REGISTRO` fossero stati salvati i dati relativi a un solo anno. Se si desidera salvare nella tabella `REGISTRO` i dati di più anni, allora si devono creare di nuovo entrambe queste viste raggruppandole in base alla colonna `Anno`. Questa colonna dovrebbe poi essere utilizzata per unirle (dal momento che la vista `TOTALEANNO` potrebbe restituire più di una riga).

Gli stessi risultati della query precedente si sarebbero potuti ottenere unendo la tabella `REGISTRO` con la vista `TOTALEANNO`, ma come si può vedere, questa query è estremamente più complicata e difficile da capire:

```
select LAST_DAY(DataAzione) MESE,
       SUM(Importo) Totale,
       (SUM(Importo)/TotaleAnno)*100 Percento
  from REGISTRO, TOTALEANNO
 where Azione = 'PAGATO'
 group by LAST_DAY(DataAzione), TotaleAnno;
```

| MESE | TOTALE | PERCENTO |
|-----------|--------|----------|
| 31-JAN-01 | 3.50 | 5.97 |
| 28-FEB-01 | 1.50 | 2.56 |
| 31-MAR-01 | 6.00 | 10.24 |
| 30-APR-01 | 9.00 | 15.36 |
| 31-MAY-01 | 3.40 | 5.80 |
| 30-JUN-01 | 4.25 | 7.25 |
| 31-JUL-01 | 4.20 | 7.17 |
| 31-AUG-01 | 11.70 | 19.96 |
| 30-SEP-01 | 4.25 | 7.25 |
| 31-OCT-01 | 4.93 | 8.41 |
| 30-NOV-01 | 3.63 | 6.19 |
| 31-DEC-01 | 2.25 | 3.84 |

12 rows selected.

Sarebbe possibile effettuare un ulteriore passo in avanti e unire semplicemente la tabella REGISTRO con se stessa, una volta per i totali mensili e un'altra per il totale annuale, senza creare alcuna vista? La risposta è negativa, perché il raggruppamento per totali mensili è in conflitto con il raggruppamento per totale annuale. Per creare query che confrontino un gruppo di righe (per mese) con un altro gruppo di righe (per anno), almeno uno dei gruppi deve essere una vista. 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.

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. Ora che esiste la vista TOTALEMESE, si può modificare la sua visualizzazione con una semplice SUBSTR (può essere utilizzata anche TO_CHAR):

```
select SUBSTR(Mese,4,3), Totale,
       (Totale/TotaleAnno)*100 Percento
  from TOTALEMESE, TOTALEANNO
 order by Mese;
SUB  TOTALE PERCENTO
--- -----
JAN   3.50    5.97
FEB   1.50    2.56
MAR   6.00    10.24
APR   9.00    15.36
MAY   3.40    5.80
JUN   4.25    7.25
JUL   4.20    7.17
AUG   11.70   19.96
SEP   4.25    7.25
OCT   4.93    8.41
NOV   3.63    6.19
DEC   2.25    3.84
```

12 rows selected.

Logica nella 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 Persona, SUM(Importo) Totale
  from REGISTRO
 where Azione = 'PAGATO'
   group by Persona
having COUNT(Articolo) > 1
  order by SUM(Importo) desc;
```

| PERSONA | TOTALE |
|---------------------------|--------|
| RICHARD KOCH AND BROTHERS | 12.00 |
| DICK JONES | 6.30 |
| ANDREW DYE | 4.50 |
| JED HOPKINS | 4.00 |
| KAY AND PALMER WALBOM | 4.00 |
| ELBERT TALBOT | 3.90 |
| WILFRED LOWELL | 3.60 |
| GERHARDT KENTGEN | 3.00 |
| PETER LAWSON | 3.00 |
| PAT LAVAY | 3.00 |
| HELEN BRANDT | 3.00 |
| VICTORIA LYNN | 2.80 |

12 rows selected.

Qui la clausola having ha selezionato solo le persone (group by ha raccolto tutte le righe in gruppi per Persona) che avevano più di una voce. Chiunque aveva effettuato un solo lavoro per Talbot è stato eliminato; quelli che hanno effettuato più di un tipo di lavoro sono stati inclusi.

Il tipo di query mostrato nell'ultimo listato è molto efficace per determinare quali righe di una tabella hanno valori duplicati in una particolare colonna. Ad 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 possibile determinare facilmente 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 seguente query mostra questo tipo di controllo effettuato per la colonna Persona della tabella REGISTRO.

```
select Persona, COUNT(*)
  from REGISTRO
   group by Persona
  having COUNT(*)>1
  order by Persona;
```

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 si 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`.

La funzione di gruppo e la colonna su cui agisce, specificate in `order by`, possono non avere alcuna relazione con le funzioni di gruppo o le colonne presenti in `select`, `group by` o `having`. Invece, se in `order by` si specifica una colonna che non fa parte di una funzione di gruppo, questa colonna deve essere necessariamente citata nella clausola `group by`. La seguente query mostra il numero di voci e l'importo totale pagato per ogni persona, ma li ordina in base all'importo medio pagato per ogni voce:

```
select Persona, COUNT(Articolo), SUM(Importo) Totale
  from REGISTRO
 where Azione = 'PAGATO'
 group by Persona
 having COUNT(Articolo) > 1
 order by AVG(Importo);
```

| PERSONA | COUNT(ARTICOLO) | TOTALE |
|---------------------------|-----------------|--------|
| PAT LAVAY | 4 | 3.00 |
| ELBERT TALBOT | 5 | 3.90 |
| GERHARDT KENTGEN | 3 | 3.00 |
| PETER LAWSON | 3 | 3.00 |
| DICK JONES | 6 | 6.30 |
| WILFRED LOWELL | 3 | 3.60 |
| VICTORIA LYNN | 2 | 2.80 |
| HELEN BRANDT | 2 | 3.00 |
| JED HOPKINS | 2 | 4.00 |
| KAY AND PALMER WALBOM | 2 | 4.00 |
| ANDREW DYE | 2 | 4.50 |
| RICHARD KOCH AND BROTHERS | 5 | 12.00 |

12 rows selected.

Se tutto ciò risulta un po' difficile da capire, si può osservare la stessa query con l'aggiunta di `AVG(Importo)` alla clausola `select`:

```
select Persona, COUNT(Articolo), SUM(Importo) Totale, AVG(Importo)
  from REGISTRO
 where Azione = 'PAGATO'
 group by Persona
 having COUNT(Articolo) > 1
 order by AVG(Importo);
```

| PERSONA | COUNT(ARTICOLO) | TOTALE | AVG(IMPORTO) |
|---------------|-----------------|--------|--------------|
| PAT LAVAY | 4 | 3.00 | .75 |
| ELBERT TALBOT | 5 | 3.90 | .78 |

| | | | |
|---------------------------|---|-------|------|
| GERHARDT KENTGEN | 3 | 3.00 | 1 |
| PETER LAWSON | 3 | 3.00 | 1 |
| DICK JONES | 6 | 6.30 | 1.05 |
| WILFRED LOWELL | 3 | 3.60 | 1.2 |
| VICTORIA LYNN | 2 | 2.80 | 1.4 |
| HELEN BRANDT | 2 | 3.00 | 1.5 |
| JED HOPKINS | 2 | 4.00 | 2 |
| KAY AND PALMER WALBOM | 2 | 4.00 | 2 |
| ANDREW DYE | 2 | 4.50 | 2.25 |
| RICHARD KOCH AND BROTHERS | 5 | 12.00 | 2.4 |

12 rows selected.

Si può controllare a mano questi risultati dal listato completo del libro mastro riportato nella Figura 10.1.

Unione di colonne

Come si è spiegato nei Capitoli 1 e 3, per unire due tabelle è necessario che queste abbiano una relazione definita da una colonna comune. Ciò vale anche se si desidera unire delle viste tra loro, oppure tabelle e liste. L'unica eccezione si ha quando una delle tabelle o delle viste ha una sola riga, come la tabella TOTALEANNO. 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.

Qualsiasi tentativo di unire due tabelle che hanno ognuna più di una colonna senza specificare le colonne di unione nella clausola where produce un *prodotto cartesiano*, di solito 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, solo poche delle quali significative.

10.4 where, having, group by e order by

Le tabelle in ORACLE possono essere raggruppate in raccolte di righe correlate, ad esempio per Articolo, per Data o per Persona. Per realizzare questo si utilizza la clausola group by, che raggruppa solo le righe della tabella che superano il test logico della clausola where:

```
where Azione='PAGATO'
group by Persona
```

La clausola having esamina questi gruppi e li elimina in base al test logico della funzione di gruppo in essa utilizzata, come:

```
having SUM(Importo) > 5
```

In questo caso vengono restituiti i gruppi per cui **SUM(Importo)** è maggiore di 5. Ogni gruppo ha solo una riga nella tabella risultante che viene visualizzata. Non è necessario (e di solito 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 SUM(Account)

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 riepilogo della tabella sottostante, che appaiono molto semplici. Una volta create, le loro colonne possono essere manipolate e le loro righe selezionate, proprio come avviene per qualsiasi altra tabella.

Queste viste possono anche essere unite fra loro e con tabelle, per produrre analisi profonde nei dati.

• Capitolo 11

• **Una query dipendente da un'altra**

• 11.1 **Sottoquery avanzate**

• 11.2 **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 opportuno andare comunque avanti. Il vantaggio è che al momento in cui questi metodi saranno necessari, si sarà in grado di utilizzarli.

11.1 **Sottoquery avanzate**

Nei Capitoli 3, 7 e 10 si sono già incontrate delle sottoquery, istruzioni select che fanno parte della clausola where di una precedente istruzione select. Le sottoquery possono essere utilizzate anche nelle istruzioni insert, update e delete, come si vedrà nel Capitolo 14.

Spesso una sottoquery fornisce un approccio alternativo a una query. Ad esempio, si supponga che George Talbot abbia urgente necessità di un trebbiatore (una persona che conduce una macchina che si sposta attraverso un campo per mietere, trebbiare e nettare cereali come grano e avena). Non può mandare nessuno in un'altra città per cercarne uno, perciò deve trovare qualcuno in Edmeston o North Edmeston. Un modo per trovare tale persona è quello di unire tre tabelle, utilizzando quelle che sono state normalizzate nel Capitolo 2 e che sono riportate nella Figura 11.1.

```
select LAVORATORE.Nome, LAVORATORE.Alloggio  
  from LAVORATORE, COMPITOLAVORATORE, ALLOGGIO  
 where LAVORATORE.Nome = COMPITOLAVORATORE.Nome  
   and LAVORATORE.Alloggio = ALLOGGIO.Alloggio  
   and Compito = 'TREBBIATORE'  
   and Address LIKE '%EDMESTON%';
```

| NOME | ALLOGGIO |
|--------------|-----------|
| JOHN PEARSON | ROSE HILL |

Tabella LAVORATORE

| NOME | ETA | ALLOGGIO |
|---------------------------|-----|------------|
| ADAH TALBOT | 23 | PAPA KING |
| ANDREW DYE | 29 | ROSE HILL |
| BART SARJEANT | 22 | CRANMER |
| DICK JONES | 18 | ROSE HILL |
| DONALD ROLLO | 16 | MATTS |
| ELBERT TALBOT | 43 | WEITBROCHT |
| GEORGE OSCAR | 41 | ROSE HILL |
| GERHARDT KENTGEN | 55 | PAPA KING |
| HELEN BRANDT | 15 | |
| JED HOPKINS | 33 | MATTS |
| JOHN PEARSON | 27 | ROSE HILL |
| KAY AND PALMER WALLBOM | | ROSE HILL |
| PAT LAVAY | 21 | ROSE HILL |
| PETER LAWSON | 25 | CRANMER |
| RICHARD KOCH AND BROTHERS | | WEITBROCHT |
| ROLAND BRANDT | 35 | MATTS |
| VICTORIA LYNN | 32 | MULLERS |
| WILFRED LOWELL | 67 | |
| WILLIAM SWING | 15 | CRANMER |

Tabella COMPITOLAVORATORE

| NOME | COMPITO | CAPACITA |
|----------------|-------------|--------------|
| Adah Talbot | Operaio | Buono |
| Dick Jones | Fabbro | Eccellente |
| Elbert Talbot | Aratore | Lento |
| Helen Brandt | Trebbiatore | Molto veloce |
| John Pearson | Trebbiatore | |
| John Pearson | Taglialegna | Buono |
| John Pearson | Fabbro | Medio |
| Victoria Lynn | Fabbro | Preciso |
| Wilfred Lowell | Operaio | Medio |
| Wilfred Lowell | Aratore | Medio |

Tabella COMPITO

| COMPITO | DESCRIZIONE |
|-------------|---|
| Trebbiatore | Badare ai cavalli, condurli, regolare le lame |
| Aratore | Badare ai cavalli, condurli, spingere arare |
| Scavatore | Segnare e aprire il terreno, scavare, punteggiare, riempire, riasettare |
| Fabbro | Accendere il fuoco, usare il soffietto, tagliare, ferrare i cavalli |
| Taglialegna | Segnare e abbattere alberi, Spaccare, Accatastare, Trasportare |
| Operaio | Lavoro generico senza compiti particolari |

Figura 11.1 Informazioni contenute nelle tabelle di Talbot. (continua)

| Tabella ALLOGGIO | | | |
|------------------|-----------------------|---------------|--------------------|
| ALLOGGIO | NOME LUNGO | DIRETTORE | INDIRIZZO |
| Cranmer | Cranmer Retreat House | Thom Cranmer | Hill St, Berkeley |
| Matts | Matts Long Bunk House | Roland Brandt | 3 Mile Rd, Keene |
| Mullers | Mullers Coed Lodging | Ken Muller | 120 Main, Edmeston |
| Papa King | Papa King Rooming | William King | 127 Main, Edmeston |
| Rose Hill | Rose Hill For Men | John Peletier | Rfd 3, N. Edmeston |
| Weitbrockt | Weitbrockt Rooming | Eunice Benson | 320 Geneva, Keene |

Figura 11.1 Informazioni contenute nelle tabelle di Talbot.

Tre tabelle vengono unite allo stesso modo di due. Le colonne comuni sono impostate come uguali fra loro nella clausola where, come mostrato nella Figura 11.2. Per unire tre tabelle insieme, occorre collegarne due a una terza.

NOTA *Non viene unita ogni tabella a ogni altra. In realtà il numero di collegamenti tra le tabelle (come LAVORATORE.Nome=COMPITOLAVORATORE.Nome) è di solito inferiore di uno rispetto al numero delle tabelle da unire).*

Una volta unite le tabelle, come è mostrato nelle prime due righe della clausola where, si possono utilizzare le colonne Compito e Indirizzo per trovare un trebbiatore nelle vicinanze.

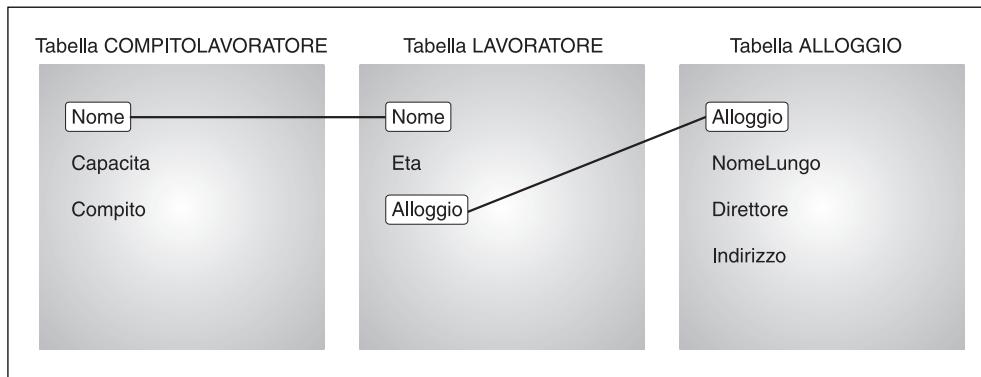


Figura 11.2 Unione di tre tabelle.

Sottoquery correlate

Esiste un altro modo per ottenere lo stesso risultato? Si ricordi che una clausola where può contenere un'istruzione select di una sottoquery. Le istruzioni select di sottoquery possono essere annidate, ovvero una clausola where in una sottoquery

può anche contenere una clausola where con una sottoquery, che a sua volta ne può contenere un'altra, fino a molti livelli, più di quanti saranno mai necessari. In genere si ritiene che i livelli siano 16.

In realtà il numero è più alto, ma non sono mai necessari neppure 16 e le prestazioni potrebbero diventare molto scadenti già dopo alcuni livelli.

Il seguente listato mostra tre istruzioni select, ciascuna connessa all'altra attraverso una clausola where:

```
select Nome, Alloggio
  from LAVORATORE

  where Nome IN
        (select Nome
           from COMPITOLAVORATORE

          where Compito = 'TREBBIATORE'
            and Alloggio IN
                  (select Alloggio
                     from ALLOGGIO

                    where Indirizzo LIKE '%EDMESTON%'));
```

| NOME | ALLOGGIO |
|--------------|-----------|
| JOHN PEARSON | ROSE HILL |

Questa query seleziona ogni trebbiatore che vive a Edmeston, semplicemente richiedendo un lavoratore il cui nome (Nome) è nella tabella COMPITOLAVORATORE con un Compito di 'TREBBIATORE' e la cui abitazione (Alloggio) è specificata nella tabella ALLOGGIO con un Indirizzo LIKE '%EDMESTON%'. Questo è un esempio di sottoquery che contiene sottoquery, ma ha anche un'altra caratteristica. Si esamini la clausola where della seconda istruzione select:

```
(select Nome
      from COMPITOLAVORATORE
     where Compito = 'TREBBIATORE'
       and Alloggio IN
```

A prima vista tutto ciò sembra sensato, ma contiene la colonna Alloggio, che non si trova nella tabella COMPITOLAVORATORE. Allora perché funziona? Il motivo è che, poiché questa è una sottoquery, ORACLE presuppone che la colonna Alloggio derivi dalla prima istruzione select, quella che contiene la sottoquery dove si trova la clausola where. Questa viene chiamata *sottoquery correlata*, perché per ogni Nome nella query esterna per cui Compito vale 'TREBBIATORE', Alloggio è correlato nella seconda clausola where.

In altre parole, una sottoquery può fare riferimento a una colonna di una tabella utilizzata nella sua query principale (la query che ha la sottoquery nella clausola where).

```
Select Nome, Alloggio
  from LAVORATORE
```

```

where Nome IN
(select Nome
 from COMPIOLAVORATORE

where Compito = 'TREBBIATORE'
and Alloggio IN
(select Alloggio
 from ALLOGGIO
 where Indirizzo LIKE ('%EDMESTON%'));
```

Questa sottoquery correlata in effetti unisce informazioni di tre tabelle per rispondere alla domanda se vi sono trebbiatori che vivono nelle vicinanze. Ma non tutte le sottoquery possono essere sostituite da unioni di tabelle. Infatti, si supponga che Talbot voglia sapere quale lavoratore è il più vecchio in ogni abitazione. Di seguito sono riportate le età massime per ogni abitazione:

```
select Alloggio, MAX(Eta)
  from LAVORATORE
 group by Alloggio;
```

| ALLOGGIO | MAX(ETA) |
|------------|----------|
| | 67 |
| CRANMER | 25 |
| MATTS | 35 |
| MULLERS | 32 |
| PAPA KING | 55 |
| ROSE HILL | 41 |
| WEITBROCHT | 43 |

Come si fa a incorporare tutto ciò in una query? La colonna Alloggio nella select superiore è correlata con la colonna Alloggio nella select della sottoquery:

```
select Nome, Eta, Alloggio
  from LAVORATORE L
 where Eta =
      (select MAX(Eta) from LAVORATORE
       where L.Alloggio = Alloggio);
```

| NOME | ETA ALLOGGIO |
|------------------|---------------|
| ELBERT TALBOT | 43 WEITBROCHT |
| GEORGE OSCAR | 41 ROSE HILL |
| GERHARDT KENTGEN | 55 PAPA KING |
| PETER LAWSON | 25 CRANMER |
| ROLAND BRANDT | 35 MATTS |
| VICTORIA LYNN | 32 MULLERS |

Occorre osservare alcune notazioni nuove. A LAVORATORE è stato assegnato un alias L e una delle colonne Alloggio nella clausola where inferiore è stata designata L.Alloggio. Questo perché la select superiore e la sottoquery utilizzano la stessa tabella e ognuna ha una colonna chiamata Alloggio.

Senza la possibilità di rinominare (ovvero assegnare un alias) a una delle tabelle, non vi sarebbe stato alcun modo per la clausola `where` inferiore di distinguere un Alloggio dall'altro. Ma avrebbe funzionato se si fosse assegnato l'alias alla seconda tabella LAVORATORE?

```
select Nome, Eta, Alloggio
  from LAVORATORE
 where Eta =
       (select MAX(Eta)
          from LAVORATORE L
         where L.Alloggio = Alloggio);
-----
```

| NOME | ETA ALLOGGIO |
|------------------|--------------|
| GERHARDT KENTGEN | 55 PAPA KING |

La risposta è ovviamente no. Il secondo `where` non comprende che Alloggio dovrebbe appartenere alla tabella LAVORATORE della query principale e perciò ritiene che `L.Alloggio` e `Alloggio` siano entrambe la sua colonna Alloggio. Dal momento che queste sono sempre uguali (eccetto che per NULL) la sottoquery non produce l'età massima per ogni abitazione, ma la massima per tutte le abitazioni.

Si richiama inoltre uno degli effetti del test di uguaglianza con NULL, discusso nel Capitolo 3: complessivamente la persona più vecchia è Wilfred Lowell, ma l'ultima query ha prodotto Gerhardt Kentgen e quella immediatamente precedente ha prodotto solo i lavoratori che hanno un valore nella colonna Alloggio (cioè quelli per cui la colonna non era NULL). Entrambi questi effetti sono dovuti a questa clausola `where`:

```
where L.Alloggio=Alloggio
```

La semplice presenza di questo test, con il segno di uguale, esclude ogni Alloggio il cui valore è NULL. Per includere i lavoratori che non hanno un valore di Alloggio, occorre utilizzare la funzione NVL:

```
select Nome, Eta, Alloggio
  from LAVORATORE L
 where Eta =
       (select MAX(Eta)
          from LAVORATORE
         where NVL(L.Alloggio,'X') = NVL(Alloggio,'X'));
```

| NOME | ETA ALLOGGIO |
|------------------|---------------|
| ELBERT TALBOT | 43 WEITBROCHT |
| GEORGE OSCAR | 41 ROSE HILL |
| GERHARDT KENTGEN | 55 PAPA KING |
| PETER LAWSON | 25 CRANMER |
| ROLAND BRANDT | 35 MATTS |
| VICTORIA LYNN | 32 MULLERS |
| WILFRED LOWELL | 67 |

Al posto di 'X' potrebbe esserci qualsiasi valore, purché sia utilizzato in entrambe le funzioni NVL. Se si aggiungesse una funzione NVL alla query che ha prodotto

solo GERHARDT KENTGEN, il risultato sarebbe WILFRED LOWELL. Un altro modo per scrivere la stessa query (e ottenere la persona più vecchia di un'abitazione) è mostrato di seguito:

```
select Nome, Eta, Alloggio
  from LAVORATORE
 where (Alloggio, Eta) IN
       (select Alloggio, MAX(Eta)
        from LAVORATORE
        group by Alloggio);
```

| NOME | ETA ALLOGGIO |
|------------------|---------------|
| PETER LAWSON | 25 CRANMER |
| ROLAND BRANDT | 35 MATTS |
| VICTORIA LYNN | 32 MULLERS |
| GERHARDT KENTGEN | 55 PAPA KING |
| GEORGE OSCAR | 41 ROSE HILL |
| ELBERT TALBOT | 43 WEITBROCHT |

Queste due colonne sono confrontate simultaneamente con una sottoquery. IN è necessario perché la sottoquery produce più di una riga, altrimenti si sarebbe dovuto utilizzare un segno di uguale. Quando due o più colonne vengono confrontate simultaneamente con il risultato di una sottoquery, devono essere racchiuse tra parentesi, come per con (Alloggio, Eta). Si noti che questa query non trova WILFRED LOWELL, perché IN ignora i valori NULL.

Coordinazione di test logici

Si supponga che John Pearson, che per molti mesi ha fatto parte della squadra di Talbot, si ammali. Chi ha le sue stesse qualifiche e dove vive? Un elenco di lavoratori conosciuti da Talbot che hanno queste qualifiche può essere facilmente ottenuto dalle tabelle LAVORATORE e COMPITOLAVORATORE:

```
select LAVORATORE.Nome, Alloggio, Compito
  from LAVORATORE, COMPITOLAVORATORE
 where LAVORATORE.Nome = COMPITOLAVORATORE.Nome;
```

| NOME | ALLOGGIO | COMPITO |
|----------------|------------|-------------|
| ADAH TALBOT | PAPA KING | OPERAIO |
| DICK JONES | ROSE HILL | FABBRO |
| ELBERT TALBOT | WEITBROCHT | ARATORE |
| HELEN BRANDT | | TREBBIATORE |
| JOHN PEARSON | ROSE HILL | TREBBIATORE |
| JOHN PEARSON | ROSE HILL | TAGLIALEGNA |
| JOHN PEARSON | ROSE HILL | FABBRO |
| VICTORIA LYNN | MULLERS | FABBRO |
| WILFRED LOWELL | | OPERAIO |
| WILFRED LOWELL | | ARATORE |

Invece di cercare manualmente in questo elenco (o nell'elenco molto più lungo che probabilmente tiene un'azienda con le qualifiche dei suoi lavoratori), si può fare svolgere questo compito a ORACLE. La query seguente chiede semplicemente quali lavoratori hanno le qualifiche di Pearson e dove vivono:

```
select LAVORATORE.Nome, Alloggio, Compito
  from LAVORATORE, COMPITOLAVORATORE
 where LAVORATORE.Nome = COMPITOLAVORATORE.Nome
   and Compito IN
      (select Compito
         from COMPITOLAVORATORE
        where Nome = 'JOHN PEARSON')
 order by LAVORATORE.Nome;
```

| NOME | ALLOGGIO | COMPITO |
|---------------|-----------|-------------|
| DICK JONES | ROSE HILL | FABBRO |
| HELEN BRANDT | | TREBBIATORE |
| JOHN PEARSON | ROSE HILL | TREBBIATORE |
| JOHN PEARSON | ROSE HILL | TAGLIALEGNA |
| JOHN PEARSON | ROSE HILL | FABBRO |
| VICTORIA LYNN | MULLERS | FABBRO |

Ovviamente nell'elenco è compreso anche Pearson; per escluderlo occorre aggiungere un altro *and* alla clausola *where*:

```
select LAVORATORE.Nome, Alloggio, Compito
  from LAVORATORE, COMPITOLAVORATORE
 where LAVORATORE.Nome = COMPITOLAVORATORE.Nome
   and Compito IN
      (select Compito
         from COMPITOLAVORATORE
        where Nome = 'JOHN PEARSON')
 and LAVORATORE.Nome != 'JOHN PEARSON'
 order by LAVORATORE.Nome;
```

| NOME | ALLOGGIO | COMPITO |
|---------------|-----------|-------------|
| DICK JONES | ROSE HILL | FABBRO |
| HELEN BRANDT | | TREBBIATORE |
| VICTORIA LYNN | MULLERS | FABBRO |

Questo *and* fa parte della query principale anche se segue la sottoquery.

EXISTS e la sua sottoquery correlata

EXISTS effettua un controllo dell'esistenza. È posto come *IN* prima di una sottoquery, ma ne differisce per due motivi:

1. non corrisponde a una colonna o a più colonne;
2. è tipicamente utilizzato con una sottoquery correlata.

Si supponga che Talbot desideri conoscere i nomi e i compiti di tutti i lavoratori che possiedono più di una qualifica. Trovare soltanto i nomi di questi lavoratori è semplice:

```
select Nome
      from COMPITOLAVORATORE
     group by Nome
    having COUNT(Compito) > 1;
```

| NOME |
|----------------|
| JOHN PEARSON |
| WILFRED LOWELL |

Tentare di trovare sia i nomi che le qualifiche non riesce, perché l'istruzione `group by`, resa necessaria da `COUNT(Compito)` è impostata sulla chiave primaria della tabella `COMPITOLAVORATORE` (`Nome, Compito`). Dal momento che per definizione ogni chiave primaria identifica univocamente soltanto una riga, il numero dei compiti per questa riga unica non può mai essere superiore a uno, perciò il test della clausola `where` risulta sempre falso e la query non trova alcuna riga:

```
select Nome, Compito
      from COMPITOLAVORATORE
     group by Nome, Compito
    having COUNT(Compito) > 1;
```

no rows selected.

`EXISTS` fornisce una soluzione. La seguente sottoquery chiede se, per ogni `Nome` selezionato nella query principale (esterna), esiste un `Nome` nella tabella `COMPITOLAVORATORE` con un numero di compiti maggiore di uno. Se per un dato nome la risposta è positiva, il test di `EXISTS` è vero e la query esterna seleziona un `Nome` e un `Compito`. I nomi dei lavoratori sono correlati dallo pseudonimo `CL` dato alla prima tabella `COMPITOLAVORATORE`.

```
select Nome, Compito
      from COMPITOLAVORATORE CL
     where EXISTS
           (select *
              from COMPITOLAVORATORE
             where CL.Nome = Nome
             group by Nome
            having COUNT(Compito) > 1);
```

| NOME | COMPITO |
|----------------|-------------|
| JOHN PEARSON | TREBBIATORE |
| JOHN PEARSON | FABBRO |
| JOHN PEARSON | TAGLIALEGNA |
| WILFRED LOWELL | OPERAIO |
| WILFRED LOWELL | ARATORE |

Questa stessa query poteva essere costruita utilizzando IN e un test sulla colonna Nome. In questo caso non è necessaria una sottoquery correlata:

```
select Nome, Compito
  from COMPITOLAVORATORE CL
 where Nome IN
   (select Nome
      from COMPITOLAVORATORE
     group by Nome
    having COUNT(Compito) > 1);
```

| NOME | COMPITO |
|----------------|-------------|
| JOHN PEARSON | TREBBIATORE |
| JOHN PEARSON | TAGLIALEGNA |
| JOHN PEARSON | FABBRO |
| WILFRED LOWELL | OPERAIO |
| WILFRED LOWELL | ARATORE |

Nell'utilizzo di EXISTS e del suo "quasi" opposto NOT EXISTS, come di ANY e ALL, vi sono alcune anomalie logiche, in particolare quando queste istruzioni sono in relazione con NULL. Queste anomalie sono trattate nella guida alfabetica di riferimento di questo libro sotto la voce EXISTS. NOT EXISTS è trattata anche nel paragrafo "Sostituzione di NOT IN con NOT EXISTS" più avanti in questo capitolo.

Unioni esterne

La tabella LAVORATORE contiene gli impiegati di Talbot e le loro età, ma non i compiti. La tabella COMPITOLAVORATORE contiene solo gli impiegati che hanno dei compiti. Come si fa a generare un report di tutti gli impiegati, con le età e i compiti, indipendentemente dal fatto che abbiano dei compiti oppure no? Si potrebbe innanzitutto provare a unire le due tabelle. Si noti l'esteso utilizzo di alias in questa query. Per convenienza ogni tabella viene rinominata in A e B; A e B appaiono ovunque al posto dei nomi delle tabelle.

```
select A.Nome, Eta, Compito
  from LAVORATORE A, COMPITOLAVORATORE B
 where A.Nome = B.Nome
 order by A.Nome;
```

| NOME | ETA | COMPITO |
|----------------|-----|-------------|
| ADAH TALBOT | 23 | OPERAIO |
| DICK JONES | 18 | FABBRO |
| ELBERT TALBOT | 43 | ARATORE |
| HELEN BRANDT | 15 | TREBBIATORE |
| JOHN PEARSON | 27 | TREBBIATORE |
| JOHN PEARSON | 27 | FABBRO |
| JOHN PEARSON | 27 | TAGLIALEGNA |
| VICTORIA LYNN | 32 | FABBRO |
| WILFRED LOWELL | 67 | OPERAIO |
| WILFRED LOWELL | 67 | ARATORE |

Sfortunatamente questo risultato include solo gli impiegati che hanno dei compiti. La soluzione per produrre un elenco completo è un'unione esterna. Questa è una tecnica con cui ORACLE aggiunge delle righe in più che non hanno corrispondenza in una tabella (in questo caso COMPITOLAVORATORE), cosicché il risultato è lungo quanto l'altra tabella (LAVORATORE). In sostanza, ogni riga della tabella LAVORATORE che non trova una corrispondenza in COMPITOLAVORATORE viene elencata comunque:

```
select A.Nome, Eta, Compito
  from LAVORATORE A, COMPITOLAVORATORE B
 where A.Nome = B.Nome(+)
 order by A.Nome;
```

| NOME | ETA | COMPITO |
|---------------------------|-----|-------------|
| ADAH TALBOT | 23 | OPERAIO |
| ANDREW DYE | 29 | |
| BART SARJEANT | 22 | |
| DICK JONES | 18 | FABBRO |
| DONALD ROLLO | 16 | |
| ELBERT TALBOT | 43 | ARATORE |
| GEORGE OSCAR | 41 | |
| GERHARDT KENTGEN | 55 | |
| HELEN BRANDT | 15 | TREBBIATORE |
| JED HOPKINS | 33 | |
| JOHN PEARSON | 27 | TREBBIATORE |
| JOHN PEARSON | 27 | TAGLIALEGNA |
| JOHN PEARSON | 27 | FABBRO |
| KAY AND PALMER WALLBOM | | |
| PAT LAVAY | 21 | |
| PETER LAWSON | 25 | |
| RICHARD KOCH AND BROTHERS | | |
| ROLAND BRANDT | 35 | |
| VICTORIA LYNN | 32 | FABBRO |
| WILFRED LOWELL | 67 | OPERAIO |
| WILFRED LOWELL | 67 | ARATORE |
| WILLIAM SWING | 15 | |

Si può vedere il segno (+), che deve trovarsi subito dopo la colonna di unione della tabella più breve, come un'indicazione ad aggiungere una riga (NULL) di B.Nome ogni volta che non vi è una corrispondenza con A.Nome. Come si può vedere, sono elencati tutti i lavoratori con l'età e il compito. Quelli per cui non vi è alcuna corrispondenza hanno la colonna Compito vuota.

Sostituzione di NOT IN con un'unione esterna

I vari test logici che possono essere effettuati in una clausola where hanno tutti differenti valori di prestazioni. Il test comunque più lento è probabilmente NOT IN, perché induce a una completa lettura della tabella nell'istruzione select della sottoquery. Si supponga che Talbot abbia necessità di lavoratori per un particolare

progetto, ma voglia essere sicuro di non chiamare quelli con qualifica di fabbro, dal momento che non sono necessari. Per selezionare i lavoratori senza la qualifica di fabbro e le loro abitazioni, potrebbe costruire una query come questa:

```
select A.Nome, Alloggio
  from LAVORATORE A
 where A.Nome NOT IN
       (select Nome
          from COMPITOLAVORATORE
         where Compito = 'FABBRO')
order by A.Nome;
```

| NOME | ALLOGGIO |
|---------------------------|------------|
| ADAH TALBOT | PAPA KING |
| ANDREW DYE | ROSE HILL |
| BART SARJEANT | CRANMER |
| DONALD ROLLO | MATTS |
| ELBERT TALBOT | WEITBROCHT |
| GEORGE OSCAR | ROSE HILL |
| GERHARDT KENTGEN | PAPA KING |
| HELEN BRANDT | |
| JED HOPKINS | MATTS |
| KAY AND PALMER WALLBOM | ROSE HILL |
| PAT LAVAY | ROSE HILL |
| PETER LAWSON | CRANMER |
| RICHARD KOCH AND BROTHERS | WEITBROCHT |
| ROLAND BRANDT | MATTS |
| WILFRED LOWELL | |
| WILLIAM SWING | CRANMER |

Questo è il modo in cui una tale query verrebbe generalmente scritta, anche se gli utenti ORACLE esperti sanno che così risulterebbe lenta. La query seguente utilizza un'unione esterna e produce lo stesso risultato, ma è molto più veloce:

```
select A.Nome, Alloggio
  from LAVORATORE A, COMPITOLAVORATORE B
 where A.Nome = B.Nome(+)
   and B.Nome is NULL
   and B.Compito(+) = 'FABBRO' order by A.Nome;
```

| NOME | ALLOGGIO |
|------------------------|------------|
| ADAH TALBOT | PAPA KING |
| ANDREW DYE | ROSE HILL |
| BART SARJEANT | CRANMER |
| DONALD ROLLO | MATTS |
| ELBERT TALBOT | WEITBROCHT |
| GEORGE OSCAR | ROSE HILL |
| GERHARDT KENTGEN | PAPA KING |
| HELEN BRANDT | |
| JED HOPKINS | MATTS |
| KAY AND PALMER WALLBOM | ROSE HILL |

| | |
|---------------------------|------------|
| PAT LAVAY | ROSE HILL |
| PETER LAWSON | CRANMER |
| RICHARD KOCH AND BROTHERS | WEITBROCHT |
| ROLAND BRANDT | MATTS |
| WILFRED LOWELL | |
| WILLIAM SWING | CRANMER |

Perché questa query funziona e dà gli stessi risultati che con NOT IN? L'unione esterna tra le due tabelle assicura che tutte le righe siano disponibili per il test, comprese quelle dei lavoratori per i quali non è elencato alcun compito nella tabella COMPITOLAVORATORE. La riga:

B.Nome is NULL

produce soltanto i lavoratori che non appaiono nella tabella COMPITOLAVORATORE (non sono elencati dei compiti). La riga:

B.Compito(+)='FABBRO'

aggiunge quelli che sono nella tabella COMPITOLAVORATORE ma non hanno una qualifica di 'FABBRO' (perciò, B.Compito(+) una qualsiasi).

La logica in questo caso è estremamente oscura, ma funziona. Il modo migliore per utilizzare questa tecnica è quello di seguire semplicemente il modello. È opportuno salvare quest'ultimo modello, inserendo una gran quantità di commenti chiificatori 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 veloce e più chiaro di effettuare questo tipo di query richiede l'utilizzo della clausola NOT EXISTS. Questa clausola viene impiegata tipicamente per determinare quali valori di una tabella non hanno valori corrispondenti in un'altra. Nell'utilizzo è identica alla clausola EXISTS; negli esempi seguenti si vedrà la differenza nella logica della query e i record restituiti.

Per ottenere dei dati da due tabelle, come COMPITO e COMPITOLAVORATORE, in genere è necessario che vengano unite. Tuttavia, l'unione di due tabelle per definizione esclude i record che esistono soltanto in una di queste. E se questi sono i record che interessano?

Ad esempio, come si fa per sapere quali qualifiche nella tabella COMPITO non sono comprese nei compiti del gruppo corrente di lavoratori? I compiti dei lavoratori sono elencati nella tabella COMPITOLAVORATORE, perciò una query che unisce COMPITO e COMPITOLAVORATORE in base alla colonna Compito escluderà le qualifiche che non sono comprese:

```
select COMPITO.Compito
      from COMPITO, COMPITOLAVORATORE
     where COMPITO.Compito = COMPITOLAVORATORE.Compito;
```

NOT EXISTS consente di utilizzare una sottoquery correlata per eliminare da una tabella tutti i record che potrebbero essere uniti a un'altra. In questo esempio, è

possibile eliminare dalla tabella COMPITO tutti i compiti che sono presenti nella colonna Compito della tabella COMPITOLAVORATORE. La seguente query mostra come si può fare:

```
select COMPITO.Compito
  from COMPITO
 where NOT EXISTS
   (select 'x' from COMPITOLAVORATORE
    where COMPITOLAVORATORE.Compito = COMPITO.Compito);

COMPITO
-----
GRAVE DIGGER
```

Come mostra questa query, non vi sono lavoratori nella tabella COMPITOLAVORATORE che hanno come compito ‘SCAVATORE’; ma come funziona?

Per ogni record nella query esterna (dalla tabella COMPITO) viene controllata la sottoquery di NOT EXISTS. Se l'unione di questo record alla tabella COMPITOLAVORATORE restituisce una riga, la sottoquery ha successo. NOT EXISTS comunica alla query di invertire il codice restituito; perciò, qualsiasi riga in COMPITO che può essere unita a COMPITOLAVORATORE non viene restituita dalla query esterna. La sola riga rimasta è l'unico compito per cui non esiste un record in COMPITOLAVORATORE.

NOT EXISTS è un modo molto efficiente per effettuare questo tipo di query, soprattutto quando per l'unione vengono utilizzate più colonne. Poiché impiega un'unione, NOT 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 significativo impatto sulle prestazioni.

11.2 UNION, INTERSECT e MINUS

Spesso è necessario combinare informazioni di tipo simile da più di una tabella. Un classico esempio potrebbero essere quello di due o più elenchi di indirizzi che debbono 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. Si supponga che Talbot abbia due elenchi, uno di propri dipendenti e un altro di potenziali lavoratori acquisito da un altro datore di lavoro. L'elenco dei dipendenti comprende i seguenti otto nomi:

```
select Nome from DIPENDENTE;
```

NOME

```
-----  
ADAH TALBOT  
DICK JONES  
DONALD ROLLO  
ELBERT TALBOT  
GEORGE OSCAR  
PAT LAVAY  
PETER LAWSON  
WILFRED LOWELL
```

L'elenco dei lavoratori potenziali comprende questi otto operai:

```
select Nome from PROSPETTIVA;
```

NOME

```
-----  
ADAH TALBOT  
DORY KENSON  
ELBERT TALBOT  
GEORGE PHEPPS  
JED HOPKINS  
PAT LAVAY  
TED BUTCHER  
WILFRED LOWELL
```

L'utilizzo più semplice di UNION è in questa combinazione delle due tabelle. Si noti che i nomi sono 12 e non 16. Quelli in entrambe le liste compaiono una sola volta:

```
select Nome from DIPENDENTE  
UNION  
select Nome from PROSPETTIVA;
```

NOME

```
-----  
ADAH TALBOT  
DICK JONES  
DONALD ROLLO  
DORY KENSON  
ELBERT TALBOT  
GEORGE OSCAR  
GEORGE PHEPPS  
JED HOPKINS  
PAT LAVAY  
PETER LAWSON  
TED BUTCHER  
WILFRED LOWELL
```

Per mostrare anche i doppioni, si può utilizzare l'operatore UNION ALL. In questo modo, qualsiasi nome presente in entrambi gli elenchi viene fornito due volte

nel risultato. Ad esempio, se si fosse utilizzato UNION ALL, la query precedente avrebbe restituito 16 nomi invece di 12.

Nel listato seguente viene effettuata l'intersezione dei due elenchi di otto lavoratori. L'elenco ottenuto contiene solo i nomi presenti in entrambe le tabelle:

```
select Nome from DIPENDENTE  
INTERSECT  
select Nome from PROSPETTIVA;
```

| | |
|----------------|-------|
| NOME | ----- |
| ADAH TALBOT | |
| ELBERT TALBOT | |
| PAT LAVAY | |
| WILFRED LOWELL | |

Poi si possono sottrarre i nomi della seconda tabella a quelli della prima. Di seguito viene sottratto PROSPETTIVA da DIPENDENTE. Restano solo i nomi che sono nella tabella DIPENDENTE e non nella tabella PROSPETTIVA:

```
select Nome from DIPENDENTE  
MINUS  
select Nome from PROSPETTIVA;
```

| | |
|--------------|-------|
| NOME | ----- |
| DICK JONES | |
| DONALD ROLLO | |
| GEORGE OSCAR | |
| PETER LAWSON | |

Questo, ovviamente, non equivale a sottrarre DIPENDENTE da PROSPETTIVA. In questo caso restano solo i nomi che si trovano in PROSPETTIVA ma non in DIPENDENTE:

```
select Nome from PROSPETTIVA  
MINUS  
select Nome from DIPENDENTE;
```

| | |
|---------------|-------|
| NOME | ----- |
| DORY KENSON | |
| GEORGE PHEPPS | |
| JED HOPKINS | |
| TED BUTCHER | |

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 si occupa solo dei nomi delle colonne ai due lati dell'operatore, ovvero richiede che ogni istruzione select sia valida e abbia colonne valide per le proprie tabelle, ma 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;
- le colonne corrispondenti devono contenere lo stesso tipo di dati (non è necessario che siano della stessa lunghezza).

La query seguente non ha senso, dal momento che prende abitazioni (Alloggio) da una tabella e nomi (Nome) dall'altra, tuttavia, poiché Alloggio e Nome sono dello stesso tipo di dati, la query funziona.

```
select Alloggio from DIPENDENTE
  UNION
select Nome from PROSPETTIVA;
```

ALLOGGIO

```
-----
ADAH TALBOT
CRANMER
DORY KENSON
ELBERT TALBOT
GEORGE PHEPPS
JED HOPKINS
MATTES
PAPA KING
PAT LAVAY
ROSE HILL
TED BUTCHER
WEITBROCHT
WILFRED LOWELL
```

Ora vengono selezionate tre colonne. Poiché DIPENDENTE è simile nella struttura alla tabella LAVORATORE, ha le colonne Nome, Alloggio ed Eta. La tabella PROSPETTIVA però ha solo le colonne Nome e Indirizzo. La seguente istruzione select confronta Alloggio e Indirizzo (dal momento che contengono informazioni simili) e aggiunge uno zero letterale all'istruzione select della tabella PROSPETTIVA in modo che corrisponda alla colonna numerica Eta nell'istruzione select di DIPENDENTE:

```
select Nome, Alloggio, Eta from DIPENDENTE
  UNION
select Nome, Address, 0 from PROSPETTIVA;
```

| NOME | ALLOGGIO | ETA |
|---------------|----------------------|-----|
| ADAH TALBOT | 23 ZWING, EDMESTON | 0 |
| ADAH TALBOT | PAPA KING | 23 |
| DICK JONES | ROSE HILL | 18 |
| DONALD ROLLO | MATTES | 16 |
| DORY KENSON | GEN. DEL., BAYBAC | 0 |
| ELBERT TALBOT | 3 MILE ROAD, WALPOLE | 0 |
| ELBERT TALBOT | WEITBROCHT | 43 |

| | | |
|----------------|--------------------|----|
| GEORGE OSCAR | ROSE HILL | 41 |
| GEORGE PHEPPS | 206 POLE, KINGSLY | 0 |
| JED HOPKINS | GEN. DEL., TURBOW | 0 |
| PAT LAVAY | 1 EASY ST, JACKSON | 0 |
| PAT LAVAY | ROSE HILL | 21 |
| PETER LAWSON | CRANMER | 25 |
| TED BUTCHER | RFD 1, BRIGHTON | 0 |
| WILFRED LOWELL | | 0 |
| WILFRED LOWELL | | 67 |

Si vede subito che molti nomi compaiono due volte. Questo perché il controllo delle ripetizioni effettuato da UNION opera sempre su tutte le colonne selezionate.

NOTA *Il controllo delle ripetizioni effettuato da UNION ignora le colonne NULL. Se nella query precedente non fosse stato inclusa la colonna Eta, WILFRED LOWELL sarebbe apparso una sola volta. Questo potrebbe non sembrare completamente logico, dal momento che NULL in una tabella non è uguale a NULL in un'altra, ma questo è il modo in cui funziona UNION.*

Applicando INTERSECT a queste stesse colonne non si ottiene alcun record, infatti nessuna riga è identica in entrambe le tabelle (che includono queste colonne). Se in questa query si fosse esclusa Eta, sarebbe stato mostrato solo WILFRED LOWELL. Anche INTERSECT ignora le colonne NULL.

```
select Nome, Alloggio, Eta from DIPENDENTE
INTERSECT
select Nome, Indirizzo, 0 from PROSPETTIVA;

no rows selected
```

Che dire di order by? Se questi operatori normalmente ordinano i risultati in base alle colonne che vengono mostrate, come può essere cambiato questo ordine? Poiché ORACLE ignora i nomi delle colonne nella combinazione di due istruzioni select, questi nomi non possono essere utilizzati in order by:

```
select Nome, Alloggio, Eta from DIPENDENTE
UNION
select Nome, Indirizzo, 0 from PROSPETTIVA
order by Eta;
*
```

ERROR at line 4: ORA-1785: order-by item must be the number of a select-list expression

order by deve essere seguito dal numero della colonna nella clausola select, ovvero dal numero che indica la posizione della colonna. Il seguente listato consente di ordinare in base a Eta, la terza colonna da sinistra:

```
select Nome, Alloggio, Eta from DIPENDENTE
UNION
select Nome, Indirizzo, 0 from PROSPETTIVA
order by 3;
```

| NOME | ALLOGGIO | ETA |
|----------------|----------------------|-----|
| ADAH TALBOT | 23 ZWING, EDMESTON | 0 |
| DORY KENSON | GEN. DEL., BAYBAC | 0 |
| ELBERT TALBOT | 3 MILE ROAD, WALPOLE | 0 |
| GEORGE PHEPPS | 206 POLE, KINGSLEY | 0 |
| JED HOPKINS | GEN. DEL., TURBOW | 0 |
| PAT LAVAY | 1 EASY ST, JACKSON | 0 |
| TED BUTCHER | RFD 1, BRIGHTON | 0 |
| WILFRED LOWELL | | 0 |
| DONALD ROLLO | MATTS | 16 |
| DICK JONES | ROSE HILL | 18 |
| PAT LAVAY | ROSE HILL | 21 |
| ADAH TALBOT | PAPA KING | 23 |
| PETER LAWSON | CRANMER | 25 |
| GEORGE OSCAR | ROSE HILL | 41 |
| ELBERT TALBOT | WEITBROCHT | 43 |
| WILFRED LOWELL | | 67 |

NOTA Questo è l'unico caso in cui si dovrebbe utilizzare il numero di posizione della colonna nella clausola *order by* in ORACLE7.1 o versioni successive. Quando non si utilizzano UNION, INTERSECT e MINUS occorre impiegare gli alias delle colonne. Si consulti *order by* nella guida alfabetica di riferimento.

Si possono utilizzare gli operatori di combinazione con due o più colonne, ma quando lo si fa, le precedenze diventano un problema, soprattutto se sono presenti INTERSECT e MINUS, perciò occorre utilizzare le parentesi per imporre l'ordine desiderato.

Sottoquery IN

Gli operatori di combinazione possono essere utilizzati nelle sottoquery, ma, con un'eccezione, hanno costrutti equivalenti che impiegano IN, AND e OR.

UNION

Nell'esempio che segue vengono cercati, nell'elenco LAVORATORE, i nomi che sono contenuti nella tabella PROSPETTIVA o in quella DIPENDENTE. Naturalmente i nomi devono trovarsi anche nella tabella LAVORATORE. Diversamente dalle combinazioni di due tabelle che non si trovano in una sottoquery, questa istruzione select è ristretta ai nomi di una sola delle tre tabelle, ovvero quelli che si trovano già nella tabella LAVORATORE:

```
select Nome
  from LAVORATORE
 where Nome IN
       (select Nome from PROSPETTIVA)
```

```
UNION  
(select Nome from DIPENDENTE);
```

NOME

ADAH TALBOT
DICK JONES
DONALD ROLLO
ELBERT TALBOT
GEORGE OSCAR
JED HOPKINS
PAT LAVAY
PETER LAWSON
WILFRED LOWELL

L'istruzione precedente sembra essere l'equivalente logica di questa:

```
select Nome  
      from LAVORATORE  
     where Nome IN  
           (select Nome from PROSPETTIVA)  
      OR Nome IN  
           (select Nome from DIPENDENTE);
```

NOME

ADAH TALBOT
DICK JONES
DONALD ROLLO
ELBERT TALBOT
GEORGE OSCAR
JED HOPKINS
PAT LAVAY
PETER LAWSON
WILFRED LOWELL

È evidente che può essere realizzato un costrutto OR equivalente a UNION.

Un avvertimento riguardo a UNION Si provi a invertire l'ordine delle tabelle a cui si applica UNION per verificare che cosa succede (gli asterischi sono stati aggiunti in seguito per evidenziare le differenze).

```
select Nome from LAVORATORE  
where Nome IN  
      (select Nome from DIPENDENTE)  
      UNION  
      (select Nome from PROSPETTIVA);
```

NOME

ADAH TALBOT
DICK JONES
DONALD ROLLO

| | |
|----------------|---|
| DORY KENSON | * |
| ELBERT TALBOT | |
| GEORGE OSCAR | |
| GEORGE PHEPPS | * |
| JED HOPKINS | |
| PAT LAVAY | |
| PETER LAWSON | |
| TED BUTCHER | * |
| WILFRED LOWELL | |

Nel risultato compaiono tre nomi che non erano nelle due precedenti versioni della query, e non sono neppure nella tabella LAVORATORE. Questo perché IN ha una precedenza maggiore rispetto a UNION. Questo significa che questo test:

```
Nome IN
(select Nome from DIPENDENTE)
```

viene valutato per primo e il risultato viene unito con questo:

```
(select Nome from PROSPETTIVA)
```

Questo risultato non è del tutto intuitivo. Se questo è il tipo di effetto che si vuole ottenere, è opportuno inserire dei commenti accanto all'istruzione SQL, perché nessuno indovinerà mai che questo era ciò che ci si aspettava. Altrimenti occorre racchiudere la serie di istruzioni select da unire tra parentesi, per fare in modo che venga elaborata per prima:

```
select Nome from LAVORATORE
where Nome IN (
    (select Nome from DIPENDENTE)
    UNION
    (select Nome from PROSPETTIVA) );
```

| NOME |
|----------------|
| ----- |
| ADAH TALBOT |
| DICK JONES |
| DONALD ROLLO |
| ELBERT TALBOT |
| GEORGE OSCAR |
| JED HOPKINS |
| PAT LAVAY |
| PETER LAWSON |
| WILFRED LOWELL |

INTERSECT

Il listato seguente cerca nella tabella LAVORATORE quei nomi che sono sia nella tabella PROSPETTIVA sia in quella DIPENDENTE. L'inversione dell'ordine delle tabelle su cui viene effettuata l'intersezione non influisce sul risultato.

```
select Nome
  from LAVORATORE
 where Nome IN
       (select Nome from PROSPETTIVA)
      INTERSECT
       (select Nome from DIPENDENTE);
```

NOME

ADAH TALBOT
ELBERT TALBOT
PAT LAVAY
WILFRED LOWELL

Il seguente listato fornisce lo stesso risultato:

```
select Nome
  from LAVORATORE
 where Nome IN
       (select Nome from PROSPETTIVA)
      AND Nome IN
       (select Nome from DIPENDENTE);
```

NOME

ADAH TALBOT
ELBERT TALBOT
PAT LAVAY
WILFRED LOWELL

MINUS

La tabella PROSPETTIVA MINUS la tabella DIPENDENTE produce soltanto un nome che è contenuto nella tabella LAVORATORE. L'inversione dell'ordine delle tabelle da sottrarre cambia i risultati, come viene mostrato nel seguito:

```
select Nome
  from LAVORATORE
 where Nome IN
       (select Nome from PROSPETTIVA)
      MINUS
       (select Nome from DIPENDENTE);
```

NOME

JED HOPKINS

Ecco la query equivalente senza MINUS:

```
select Nome
  from LAVORATORE
```

```
where Nome IN
      (select Nome from PROSPETTIVA)
and Nome NOT IN
      (select Nome from DIPENDENTE);
```

NOME

JED HOPKINS

Invertendo l'ordine delle tabelle si ottengono questi risultati:

```
select Nome from LAVORATORE
where Nome IN
      (select Nome from DIPENDENTE)
      MINUS
      (select Nome from PROSPETTIVA);
```

NOME

DICK JONES
DONALD ROLLO
GEORGE OSCAR
PETER LAWSON

Questo è un risultato molto più intuitivo dell'inversione delle tabelle in UNION, perché dipende da quale tabella viene sottratta all'altra (si rimanda al paragrafo “UNION, INTERSECT e MINUS” per maggiori dettagli) e la precedenza di IN non influisce.

Un avvertimento riguardo a MINUS Non è insolito utilizzare MINUS quando una delle tabelle nella sottoquery è la stessa tabella della query principale. Il pericolo è che questa, se è la tabella dopo MINUS, venga sottratta da se stessa. Senza un altro qualificatore (come nella clausola where della select che segue MINUS), non viene selezionato alcun record.

```
select Nome
      from PROSPETTIVA
     where Nome IN
           (select Nome from DIPENDENTE)
           MINUS
           (select Nome from PROSPETTIVA);
```

no rows selected

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:

```
select Nome from LAVORATORE
where (Nome, Alloggio) IN
```

```
(select Nome, Alloggio from DIPENDENTE)
    MINUS
    (select Nome, Indirizzo from PROSPETTIVA);
```

ERROR at line 1: ORA-1789: query block has incorrect number of result columns

Ma una costruzione IN equivalente non ha queste limitazioni:

```
select Nome from LAVORATORE
where (Nome, Alloggio) IN
    (select Nome, Alloggio from DIPENDENTE)
AND (Nome, Alloggio) NOT IN
    (select Nome, Address from PROSPETTIVA);
```

| NOME |
|---------------|
| ADAH TALBOT |
| DICK JONES |
| DONALD ROLLO |
| ELBERT TALBOT |
| GEORGE OSCAR |
| PAT LAVAY |
| PETER LAWSON |

Per fare in modo che la versione con MINUS funzioni, tutte le colonne nella clausola where devono essere nella clausola select. Devono anche essere, naturalmente, nelle istruzioni select a cui è applicato MINUS:

```
select Nome, Alloggio from LAVORATORE
where (Nome, Alloggio) IN
    (select Nome, Alloggio from DIPENDENTE)
    MINUS
    (select Nome, Address from PROSPETTIVA);
```

| NOME | ALLOGGIO |
|---------------|------------|
| ADAH TALBOT | PAPA KING |
| DICK JONES | ROSE HILL |
| DONALD ROLLO | MATTS |
| ELBERT TALBOT | WEITBROCHT |
| GEORGE OSCAR | ROSE HILL |
| PAT LAVAY | ROSE HILL |
| PETER LAWSON | CRANMER |

Alcuni libri e altre pubblicazioni sostengono che non è possibile utilizzare operatori di combinazione in sottoquery, ma non è vero. Ecco un esempio di come usarli:

```
select Nome from LAVORATORE
where (Nome, Alloggio) IN
    (select Nome, Alloggio from LAVORATORE
        where (Nome, Alloggio) IN
```

```
(select Nome, Address from PROSPETTIVA)
  UNION
  (select Nome, Alloggio from DIPENDENTE) );
```

NOME

```
-----  
ADAH TALBOT  
DICK JONES  
DONALD ROLLO  
ELBERT TALBOT  
GEORGE OSCAR  
PAT LAVAY  
PETER LAWSON
```

L'utilizzo 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.

•
•
•
•
• Capitolo 12

• **Alcune possibilità complesse**

• 12.1 **Creazione di una vista complessa**

• 12.2 **Alberi genealogici e connect by**

• 12.3 **Utilizzo di viste nella clausola from**

•
•
•
•

Questo capitolo continua lo studio delle funzioni e delle caratteristiche più complesse di ORACLE. Di particolare interesse è la creazione di semplici query 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 11, 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 se necessario.

12.1 Creazione di una vista complessa

È possibile costruire viste basate l'una sull'altra. Nel Capitolo 10 è stato introdotto il concetto di creazione di una vista di un raggruppamento di righe da una tabella. Qui questo concetto viene esteso per mostrare come le viste possano essere unite a altre viste e tabelle per produrre un'altra vista. Questa tecnica, anche se sembra un po' complicata, in realtà semplifica la realizzazione di query e report. Di seguito è riportato un elenco delle spese sostenute da G.B. Talbot nel marzo del 1901. Fu un mese difficile. Voci, nomi e importi sono presi direttamente dal libro mastro.

```
column Importo format 999.90
column Articolo format a23
column Persona format a16 column

select DataAzione, Articolo, Persona, Importo
  from REGISTRO
 where DataAzione BETWEEN
       TO_DATE('01-MAR-1901','DD-MON-YYYY') and
       TO_DATE('31-MAR-1901','DD-MON-YYYY')
   and Azione IN ('COMPRATO','PAGATO')
 order by DataAzione;
```

| DATAAZIONE | ARTICOLO | PERSONA | IMPORTO |
|------------|-----------------------|------------------|---------|
| 05-MAR-01 | TELEFONATA | PHONE COMPANY | .20 |
| 06-MAR-01 | MEDICINE INDIGESTIONE | DR. CARLSTROM | .40 |
| 06-MAR-01 | PANTALONI | GENERAL STORE | .75 |
| 07-MAR-01 | SCARPE | BLACKSMITH | .35 |
| 07-MAR-01 | POSTA | POST OFFICE | 1.00 |
| 08-MAR-01 | TABACCO DA MASTICARE | MILL | .25 |
| 10-MAR-01 | MANICOTTO TUBO STUFA | VERNA HARDWARE | 1.00 |
| 13-MAR-01 | TERMOMETRO | GENERAL STORE | .15 |
| 14-MAR-01 | LOTTO CIMITERO N. 80 | METHODIST CHURCH | 25.00 |
| 14-MAR-01 | SCAVO | JED HOPKINS | 3.00 |
| 16-MAR-01 | MACINA | MILL | .16 |
| 20-MAR-01 | LAVORO | DICK JONES | 1.00 |
| 22-MAR-01 | CONTENITORI LATTE | VERNA HARDWARE | 5.00 |
| 23-MAR-01 | FODERE VESTITI | GENERAL STORE | .54 |
| 25-MAR-01 | STIVALI PER SHIRLEY | GENERAL STORE | 2.50 |
| 27-MAR-01 | ZUPPA VERDURA | MILL | .77 |
| 30-MAR-01 | RIPARAZIONE OROLOGIO | MANNER JEWELERS | .25 |

Se riordinate per Persona, si può vedere come sono state concentrate le spese di Talbot:

```
select DataAzione, Articolo, Persona, Importo
  from REGISTRO
 where DataAzione BETWEEN
      TO_DATE('01-MAR-1901','DD-MON-YYYY') and
      TO_DATE('31-MAR-1901','DD-MON-YYYY')
    and Azione IN ('COMPRATO','PAGATO')
  order by Persona, DataAzione;
```

| DATAAZIONE | ARTICOLO | PERSONA | IMPORTO |
|------------|-----------------------|------------------|---------|
| 07-MAR-01 | SCARPE | BLACKSMITH | .35 |
| 20-MAR-01 | LAVORO | DICK JONES | 1.00 |
| 06-MAR-01 | MEDICINE INDIGESTIONE | DR. CARLSTROM | .40 |
| 06-MAR-01 | PANTALONI | GENERAL STORE | .75 |
| 13-MAR-01 | TERMOMETRO | GENERAL STORE | .15 |
| 23-MAR-01 | FODERE VESTITI | GENERAL STORE | .54 |
| 25-MAR-01 | STIVALI PER SHIRLEY | GENERAL STORE | 2.50 |
| 14-MAR-01 | SCAVO FOSSA | JED HOPKINS | 3.00 |
| 30-MAR-01 | RIPARAZIONE OROLOGIO | MANNER JEWELERS | .25 |
| 14-MAR-01 | LOTTO CIMITERO N. 80 | METHODIST CHURCH | 25.00 |
| 08-MAR-01 | TABACCO DA MASTICARE | MILL | .25 |
| 16-MAR-01 | MACINA | MILL | .16 |
| 27-MAR-01 | ZUPPA VERDURA | MILL | .77 |
| 05-MAR-01 | TELEFONATA | PHONE COMPANY | .20 |
| 07-MAR-01 | POSTA | POST OFFICE | 1.00 |
| 10-MAR-01 | MANICOTTO TUBO STUFA | VERNA HARDWARE | 1.00 |
| 22-MAR-01 | CONTENITORI LATTE | VERNA HARDWARE | 5.00 |

Una vista di un gruppo

Da questa tabella viene creata una vista, raggruppata per Persona, cosicché sia possibile vedere quanto Talbot ha speso per ogni fornitore. Si noti l'utilizzo di alias per la colonna calcolata:

```
create or replace view TOTALEARTICOLO as
select Persona, SUM(Importo) TotaleArticolo
  from REGISTRO
 where DataAzione BETWEEN
      TO_DATE('01-MAR-1901','DD-MON-YYYY') and
      TO_DATE('31-MAR-1901','DD-MON-YYYY')
    and Azione IN ('COMPRATO','PAGATO')
 group by Persona;
```

View created.

Qui è riportato quello che contiene la vista:

```
column TotaleArticolo format 99,999.90
```

```
select * from TOTALEARTICOLO;
```

| PERSONA | TOTALEARTICOLO |
|------------------|----------------|
| BLACKSMITH | .35 |
| DICK JONES | 1.00 |
| DR. CARLSTROM | .40 |
| GENERAL STORE | 3.94 |
| JED HOPKINS | 3.00 |
| MANNER JEWELERS | .25 |
| METHODIST CHURCH | 25.00 |
| MILL | 1.18 |
| PHONE COMPANY | .20 |
| POST OFFICE | 1.00 |
| VERNA HARDWARE | 6.00 |

Una vista del totale

Poi viene creata un'altra vista con esattamente le stesse informazioni, ma senza la clausola group by. In questo modo si ottiene un totale completo di tutti i record:

```
create or replace view TOTALE as
select SUM(Importo) TOTALE
  from REGISTRO
 where DataAzione BETWEEN
      TO_DATE('01-MAR-1901','DD-MON-YYYY') and
      TO_DATE('31-MAR-1901','DD-MON-YYYY')
    and Azione IN ('COMPRATO','PAGATO');
```

View created.

Questa lista contiene soltanto un record:

```
select * from TOTALE;
```

```
TOTALE
-----
42.32
```

La vista combinata

Infine viene creata un'altra vista, che contiene la tabella di base, REGISTRO, la vista dei totali per voci TOTALEARTICOLO, la vista della spesa totale per tutte le voci TOTALE. Si noti di nuovo notare l'utilizzo di alias per le colonne calcolate e per i nomi delle tabelle e viste sottostanti. Questa vista è in effetti un'unione in tre vie di una tabella con se stessa, utilizzando viste che riassumono la tabella in due maniere diverse:

```
create or replace view PerArticolo as
select L.Persona Persona, Articolo, Importo,
       100*Importo/TotaleArticolo PerPersona, 100*Importo/Totale PerTotale
  from REGISTRO L, TOTALEARTICOLO I, TOTALE
 where L.PERSONA = I.PERSONA
   and DataAzione BETWEEN
      TO_DATE('01-MAR-1901','DD-MON-YYYY') and
      TO_DATE('31-MAR-1901','DD-MON-YYYY')
   and Azione IN ('COMPRATO','PAGATO');
```

View created.

Per realizzare le somme vengono utilizzate tre istruzioni compute sum, insieme con una break on. Si noti la semplicità dell'istruzione select in questo caso:

```
column PerPersona format 9,999.99
column PerTotale format 9,999.99

break on Persona skip 1
compute sum of PerPersona on Persona
compute sum of PerTotal on Persona
compute sum of Importo on Persona

select Persona, Articolo, Importo, PerPersona, PerTotal
  from PerArticolo
 order by Persona;
```

E ora si deve esaminare l'enorme quantità di informazioni prodotta. Non soltanto viene mostrata ogni voce con il relativo prezzo, ma sulla stessa riga vengono presentate anche la percentuale di tutte le spese per quella Persona e la percentuale di tutte le spese totali:

| PERSONA | ARTICOLO | IMPORTO | PER PERSONA | PERTOTALE |
|------------------|-----------------------|---------|-------------|-----------|
| BLACKSMITH | SCARPE | .35 | 100.00 | .83 |
| ***** | | | | |
| sum | | .35 | 100.00 | .83 |
| DICK JONES | LAVORO | 1.00 | 100.00 | 2.36 |
| ***** | | | | |
| sum | | 1.00 | 100.00 | 2.36 |
| DR. CARLSTROM | MEDICINE INDIGESTIONE | .40 | 100.00 | .95 |
| ***** | | | | |
| sum | | .40 | 100.00 | .95 |
| GENERAL STORE | PANTALONI | .75 | 19.04 | 1.77 |
| | FODERE VESTITI | .54 | 13.71 | 1.28 |
| | TERMOMETRO | .15 | 3.81 | .35 |
| | STIVALI PER SHIRLEY | 2.50 | 63.45 | 5.91 |
| ***** | | | | |
| sum | | 3.94 | 100.00 | 9.31 |
| JED HOPKINS | SCAVO FOSSA | 3.00 | 100.00 | 7.09 |
| ***** | | | | |
| sum | | 3.00 | 100.00 | 7.09 |
| MANNER JEWELERS | RIPARAZIONE OROLOGIO | .25 | 100.00 | .59 |
| ***** | | | | |
| sum | | .25 | 100.00 | .59 |
| METHODIST CHURCH | LOTTO CIMITERO N. 80 | 25.00 | 100.00 | 59.07 |
| ***** | | | | |
| sum | | 25.00 | 100.00 | 59.07 |
| MILL | TABACCO PER BOCCA | .25 | 21.19 | .59 |
| | ZUPPA VERDURA | .77 | 65.25 | 1.82 |
| | MACINA | .16 | 13.56 | .38 |
| ***** | | | | |
| sum | | 1.18 | 100.00 | 2.79 |
| PHONE COMPANY | TELEFONATA | .20 | 100.00 | .47 |
| ***** | | | | |
| sum | | .20 | 100.00 | .47 |
| POST OFFICE | POSTA | 1.00 | 100.00 | 2.36 |
| ***** | | | | |
| sum | | 1.00 | 100.00 | 2.36 |
| VERNA HARDWARE | MANICOTTO TUBO STUFA | 1.00 | 16.67 | 2.36 |
| | CONTENITORI LATTE | 5.00 | 83.33 | 11.81 |
| ***** | | | | |
| sum | | 6.00 | 100.00 | 14.18 |

Con questa tecnica, che utilizza viste di riepilogo di una tabella unita a se stessa e viste di varie tabelle unite insieme, si possono creare viste e report che includono medie ponderate, rendimento effettivo, percentuale del totale, percentuale di totali parziali e molti calcoli simili. Non vi è 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. Il comando break on report, che è stato utilizzato per i totali generali, è trattato nel Capitolo 13.

12.2 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 un mezzo 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 è tecnicamente chiamato *nodo radice*. Nel linguaggio corrente corrisponderebbe al tronco. Dal tronco si estendono i rami, che hanno altri rami, che hanno ancora altri rami. Le forcille in cui uno o più rami si distaccano da un ramo più largo sono chiamati *nodi* e l'estremità di un ramo è chiamata *foglia* o *nodo foglia*. La Figura 12.1 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 12.1, si scopre che corrispondono. EVE non ha alcuna mucca o toro genitore, perché è della prima generazione, e ADAM e BANDIT sono tori introdotti per la riproduzione, senza genitori nella tabella.

```

column Mucca format a6
column Toro format a6
column Figlio format a10
column Sex format a3

select * from ALLEVAMENTO
order by Datanascita;

FIGLIO      SEX MUCCA    TORO      DATANASCITA
-----  -----  -----
EVE          F
ADAM         M
BANDIT       M

```

| | | | | |
|-------|---|-------|--------|-----------|
| BETSY | F | EVE | ADAM | 02-JAN-00 |
| POCO | M | EVE | ADAM | 15-JUL-00 |
| GRETA | F | EVE | BANDIT | 12-MAR-01 |
| 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 |

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 tenuta da Talbot, DELLA è la quinta generazione. Ogniqualvolta viene impiegata la clausola connect by, la colonna Level può essere aggiunta all'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 che segue, ma perché l'istruzione select ha prodotto questo? Come funziona?

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sex, DataNascita
  from ALLEVAMENTO
 where Figlio = 'EVE'
 connect by Mucca = PRIOR Figlio;
```

| MUCCA | TORO | FIGLIO | SEX | DATANASCITA |
|-------|--------|--------|-------|-------------|
| <hr/> | | | | |
| EVE | | | | |
| EVE | ADAM | BETSY | 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 | 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 |

Si noti che questa è in pratica la Figura 12.1 percorsa in senso orario. EVE non è al centro, ma è il nodo radice (tronco) di questo albero. I suoi figli sono BETSY, POCO, GRETA, MANDY e CINDY.

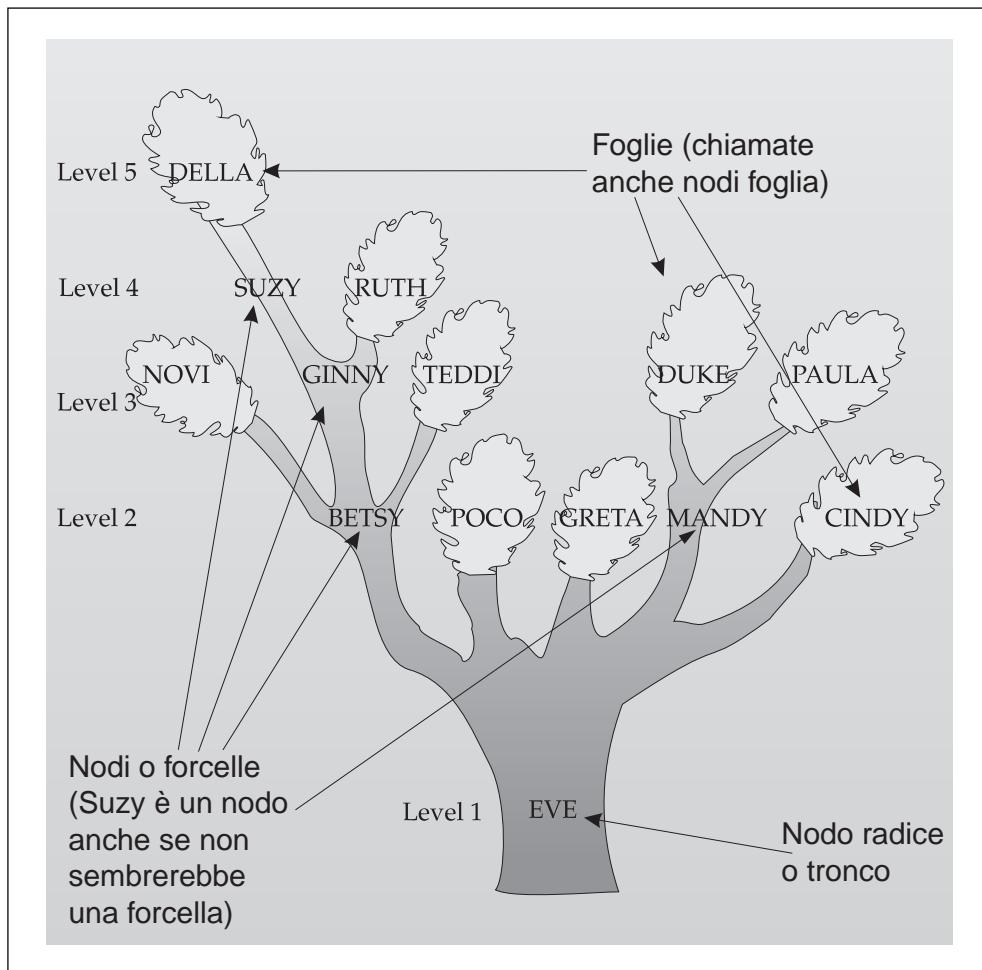


Figura 12.1 Le mucche e i tori di Talbot, con la capostipite EVE.

I figli di BETSY sono NOVI, GINNY e TEDDI. I figli di GINNY sono SUZY e RUTH e la figlia di SUZY è DELLA. Anche MANDY ha due figli, che si chiamano DUKE e PAULA.

Questo albero inizia con EVE come primo “descendente”. 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 una costruzione molto confusa. Il formato di LPAD, riportato dal Capitolo 6, è il seguente:

```
LPAD(stringa, lunghezza[, 'set'])
```

Occorre confrontarlo con LPAD nell'istruzione select mostrata in precedenza:

```
LPAD(' ',6*(Level-1))
```

I due apici singoli, separati da uno spazio, non definiscono il carattere da utilizzare per il riempimento. È una colonna letterale, una stringa (*stringa*) lunga un carattere. 6*(Level-1) è la lunghezza (*lunghezza*) e poiché *set* non è definito, viene utilizzato il valore di default, uno spazio. 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). Qual è questo numero?

6*(Level-1)

Il calcolo è svolto dapprima sottraendo 1 da Level (per EVE Level è 1, quindi 1-1 è 0; per BETSY Level (la generazione) è 2, perciò 2-1 è 1); quindi questo risultato viene moltiplicato per 6 e il numero che si ottiene indica quanti spazi devono essere concatenati alla sinistra della colonna Figlio. Occorre notare che LPAD non riempie Figlio direttamente, ma vi è concatenato. L'effetto è evidente nel risultato appena mostrato. Ogni generazione, o livello, è riempita con spazi proporzionali a Level.

Perché si effettuano un riempimento e una concatenazione, invece di utilizzare semplicemente LPAD direttamente su Figlio? Per due motivi. Innanzitutto, un'istruzione LPAD diretta su Figlio avrebbe allineato i nomi a destra, per cui i nomi di ogni livello avrebbero avuto l'ultima lettera allineata verticalmente. In secondo luogo, se Level-1 è uguale a 0, come per EVE, il risultato di LPAD sarebbe stato di 0 caratteri, ovvero EVE sarebbe sparita:

```
select Mucca, Toro, LPAD(Figlio,6*(Level-1),' ') Figlio,
       Sex, Datanascita from ALLEVAMENTO
  start with Figlio = 'EVE'
 connect by Mucca = PRIOR Figlio;
```

| MUCCA | TORO | FIGLIO | SEX | 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 |
| | | | F | 09-FEB-03 |

Perciò, 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 su Figlio.

Ma come funziona connect by? Si esamini di nuovo la Figura 12.1. A partire da NOVI e muovendosi all'indietro, quali mucche sono i discendenti 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 colonna 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 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,
       Sex, Datanascita
  from ALLEVAMENTO
 start with Figlio = 'EVE'
 connect by Mucca = PRIOR Figlio
        and Figlio != 'BETSY';
```

| MUCCA | TORO | FIGLIO | SEX | 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 realtà occorre notare che BETSY è ancora lì, sotto la colonna Mucca come madre dei suoi figli, NOVI, GINNY e TEDDI:

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sex, Datanascita
  from ALLEVAMENTO
 where Figlio != 'BETSY'
 start with Figlio = 'EVE'
 connect by Mucca = PRIOR Figlio;
```

| MUCCA | TORO | FIGLIO | | SEX | 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 utilizzando connect by è livello per livello, da sinistra a destra, come mostrato nella Figura 12.1, iniziando con il livello più basso. Nell'esempio seguente si vuole alterare questo ordine in modo da raggruppare le mucche e i figli in base alla data di nascita:

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sex, Datanascita   from ALLEVAMENTO
  start with Figlio = 'EVE'
connect by Mucca = PRIOR Figlio
  order by Mucca, Datanascita;
```

| MUCCA | TORO | FIGLIO | | SEX | DATANASCITA |
|-------|--------|--------|--|-----|-------------|
| | | EVE | | F | |
| 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 |

Le generazioni sono ancora evidenti nella visualizzazione, ma i discendenti sono raggruppati più vicini in base alla loro madre. Lo stesso albero genealogico si può esaminare anche per Datanascita, come segue:

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sex, Datanascita
  from ALLEVAMENTO
  start with Figlio = 'EVE'
```

```
connect by Mucca = PRIOR Figlio
order by Datanascita;
```

| MUCCA | ORO | FIGLIO | SEX | DATANASCITA |
|-------|--------|--------|-----|-------------|
| EVE | ADAM | BETSY | F | |
| 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 |

Ora l'ordine delle righe non mostra più le generazioni, come in un albero, ma i rientri ancora conservano questa informazione. Però non è possibile stabilire la relazione tra genitori e discendenti senza osservare le colonne Mucca e Toro.

Spostarsi verso le radici

In precedenza 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,
      Sex, Datanascita
     from ALLEVAMENTO
    start with Figlio = 'DELLA'
 connect by Figlio = PRIOR Mucca;
```

| MUCCA | TORO | FIGLIO | SEX | 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. Sembra che DELLA sia l'antenata ed EVE l'ultima discendente.

Aggiungere un order by Datanascita può essere d'aiuto, ma EVE è ancora all'estrema destra:

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
       Sex, Datanascita
      from ALLEVAMENTO
    start with Figlio = 'DELLA'
connect by Figlio = PRIOR Mucca
      order by Datanascita;
```

| MUCCA | TORO | FIGLIO | SEX | DATANASCITA |
|-------|--------------|--------|-----|-------------|
| EVE | ADAM | BETSY | EVE | F |
| BETSY | BANDIT | GINNY | | 02-JAN-00 |
| GINNY | DUKE | SUZY | | 04-DEC-03 |
| SUZY | BANDIT DELLA | | | 03-APR-06 |
| | | | | 11-OCT-08 |

La soluzione consiste semplicemente nel cambiare il calcolo in LPAD:

```
select Mucca, Toro, LPAD(' ',6*(5-Level))||Figlio Figlio,
       Sex, Datanascita
      from ALLEVAMENTO
    start with Figlio = 'DELLA'
connect by Figlio = PRIOR Mucca
      order by Datanascita;
```

| MUCCA | TORO | FIGLIO | SEX | DATANASCITA |
|-------|--------|--------|-----|-------------|
| EVE | ADAM | BETSY | EVE | F |
| BETSY | BANDIT | GINNY | | 02-JAN-00 |
| GINNY | DUKE | SUZY | | 04-DEC-03 |
| SUZY | BANDIT | DELLA | | 03-APR-06 |
| | | | | 11-OCT-08 |

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,
       Sex, Datanascita
      from ALLEVAMENTO
    start with Figlio = 'ADAM'
connect by PRIOR Figlio = Toro;
```

| MUCCA | TORO | FIGLIO | SEX | DATANASCITA |
|-------|------|--------|------|-------------|
| EVE | ADAM | BETSY | ADAM | M |
| EVE | ADAM | POCO | | 02-JAN-00 |
| EVE | POCO | MANDY | | 15-JUL-00 |
| EVE | POCO | CINDY | | 22-AUG-02 |
| MANDY | POCO | PAULA | | 09-FEB-03 |
| BETSY | ADAM | NOVI | | 21-DEC-06 |
| | | | | 30-MAR-03 |

ADAM e BANDIT erano i tori originari che hanno dato inizio alla mandria. Per creare un singolo albero che riporta i discendenti sia di ADAM sia di BANDIT, si dovrebbe inventare un “padre” per i due, 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 in più di un modo:

```
select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
      Sex, Datanascita
     from ALLEVAMENTO
   start with Figlio = 'BANDIT'
 connect by PRIOR Figlio = Toro;
```

| MUCCA | TORO | FIGLIO | SEX | DATANASCITA |
|-------|--------|--------|-----|-------------|
| | BANDIT | | M | |
| EVE | BANDIT | GRETA | F | 12-MAR-01 |
| BETSY | BANDIT | GINNY | F | 04-DEC-03 |
| MANDY | BANDIT | DUKE | M | 24-JUL-04 |
| GINNY | DUKE | SUZY | F | 03-APR-06 |
| GINNY | DUKE | RUTH | F | 25-DEC-06 |
| BETSY | BANDIT | TEDDI | F | 12-AUG-05 |
| SUZY | BANDIT | DELLA | 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 impiega 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 individui dall’albero, ma non i 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 i suoi discendenti (o antenati, a seconda di come si traccia l’albero).
- connect by non può essere utilizzata con un’unione di tabelle nella clausola where.

Questa particolare serie di comandi probabilmente viene ricordata in modo esatto da poche persone, ma con una conoscenza di 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.

12.3 Utilizzo di viste nella clausola from

Quando si scrive una query che unisce una tabella a una vista, quest'ultima deve già esistere. Se la vista è utilizzata solo per quella query, è possibile crearla nella query stessa. La capacità di creare una vista durante l'esecuzione della query è disponibile a partire da ORACLE7.2.

In precedenza in questo capitolo è stata creata la vista TOTALE per calcolare la somma di REGISTRO.Importo per particolari valori di DataAzione. Il seguente elenco mostra la sintassi per la vista TOTALE:

```
create or replace view TOTALE as
select SUM(Importo) TOTALE
  from REGISTRO
 where DataAzione BETWEEN
       TO_DATE('01-MAR-1901','DD-MON-YYYY') and
       TO_DATE('31-MAR-1901','DD-MON-YYYY')
  and Azione IN ('COMPRATO','PAGATO');
```

La vista TOTALE può essere unita a REGISTRO per mostrare il valore percentuale con cui ogni Importo contribuisce alla somma totale. La seguente query unisce REGISTRO a TOTALE e applica le stesse condizioni per le clausole where della tabella REGISTRO e della vista TOTALE:

```
select Persona, Importo, 100*Importo/Totale
  from REGISTRO, TOTALE
 where DataAzione BETWEEN
       TO_DATE('01-MAR-1901','DD-MON-YYYY') AND
       TO_DATE('31-MAR-1901','DD-MON-YYYY')
  and Azione IN ('COMPRATO','PAGATO');
```

Quando viene eseguita la query precedente, ORACLE determina l'output della vista (il singolo riepilogo di riga della colonna Importo) e utilizza questo valore durante la risoluzione del resto della query.

A partire da ORACLE7.2, la sintassi della vista può essere inserita direttamente nella clausola from della query, perciò non è necessario creare la vista TOTALE. Il seguente elenco mostra la query combinata. In essa, il codice SQL della vista TOTALE è inserito come una sottoquery nella clausola from. La colonna Totale della vista viene utilizzata nell'elenco delle colonne nella query principale. La query combinata è funzionalmente identica alla query che impiegava la vista TOTALE.

```
select Persona, Importo, 100*Importo/Totale
  from REGISTRO,
       (select SUM(Importo) TOTALE
        from REGISTRO
       where DataAzione BETWEEN
             TO_DATE('01-MAR-1901','DD-MON-YYYY') and
```

```
        TO_DATE('31-MAR-1901','DD-MON-YYYY')
        and Azione IN ('COMPRATO','PAGATO') )
where DataAzione BETWEEN
        TO_DATE('01-MAR-1901','DD-MON-YYYY') and
        TO_DATE('31-MAR-1901','DD-MON-YYYY')
and Azione IN ('COMPRATO','PAGATO');
```

Il vantaggio del metodo integrato è che non vi è alcuna necessità di creare e conservare la vista TOTALE, perché è contenuta nella query che richiede.

Le sottoquery possono essere complesse. Se è necessario unire colonne di sottoquery con colonne di altre tabelle, occorre semplicemente assegnare a ogni colonna un alias univoco e fare riferimento a questi alias nelle unioni nella clausola where della query.

• Capitolo 13

• Creazione di un report in SQLPLUS

- 13.1 Formattazione avanzata
- 13.2 set termout off e set termout on
- 13.3 Variabili in SQLPLUS
- 13.4 Formattazione di numeri
- 13.5 Utilizzo di mask.sql
- 13.6 Utilizzo di buffer per salvare comandi
SQLPLUS
- 13.7 show all e spool
- 13.8 Aggiunta di righe vuote
- 13.9 Ulteriori controlli per la realizzazione
di report

Nel Capitolo 5 sono stati descritti i concetti fondamentali per la formattazione di report e nei Capitoli 3 e 10 sono stati forniti metodi per l'utilizzo di gruppi e viste. In questo capitolo vengono esaminati metodi più avanzati di formattazione e calcoli più complessi di medie ponderate. Un certo numero di interdipendenze fra i vari comandi SQLPLUS non è documentato in alcun manuale o libro su ORACLE; in questo capitolo ne viene presentata una rassegna.

13.1 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
```

La prima istruzione select estrae le azioni delle tre industrie, esegue un semplice calcolo della differenza nei prezzi di chiusura fra oggi e ieri e le ordina per Industria e Società, come illustrato nella Figura 13.1.

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,
```

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 ampio report, ad 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
```

```
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 13.1 Prezzi e volume di chiusura delle azioni.

Problemi di formattazione con i numeri

Si noti che saltuariamente si possono verificare dei problemi con i numeri nelle colonne le cui intestazioni sono più grandi delle definizioni di formattazione. Ad esempio, una colonna chiamata Interesse con un formato 99.90 ha nove caratteri nel nome ma solo quattro cifre di spazio assegnato.

SQLPLUS prende le informazioni di formattazione per i calcoli interni dal formato del numero e non dall'intestazione della colonna. Perciò, una colonna con un nome che è più lungo del numero totale delle cifre a volte viene visualizzata o stampata in SQLPLUS con caratteri strani. La soluzione è quella di fare in modo che il numero di cifre nell'istruzione di formato sia sempre pari almeno alla lunghezza del titolo della colonna, meno uno (per il segno meno).

```
Interesse
-----
9999.99
```

break on

Poi si imposta break on per inserire una riga vuota tra la fine di un gruppo Industria e l'inizio del successivo; in seguito viene calcolata la somma del volume giornaliero per ogni industria. Si osservi nella Figura 13.2 la coordinazione fra break on e compute sum.

Ora break on e compute sum vengono estese. L'ordine delle colonne in break on è critico, come verrà spiegato tra poco. compute sum è stata incaricata di calcolare Volume per Industria e Report (Figura 13.3).

Il volume viene mostrato per ogni industria come in precedenza, ma ora è stato aggiunto anche un volume totale per tutte le industrie. Il comando compute è stato esteso per consentire il calcolo su tutto il report. Questo comando deve essere coordinato con break on:

```
break on Report on Industria skip 1
compute sum of Volume on Industria Report
```

Questo codice, che riporta i comandi break on e compute nella Figura 13.3, produce una somma di Volume per l'intero report. Il posizionamento di on Report nel comando break on non ha importanza: on Report viene sempre calcolato alla fine.

Poi al report viene assegnato un titolo in alto e uno in fondo, utilizzando il nuovo metodo per ttitle e btitle che consente una formattazione estesa. Per una spiegazione di questi comandi occorre consultare il paragrafo “Comandi di formattazione ttitle e btitle”.

```
ttitle left 'Portfolio corrente' -
      right '1 aprile 1998'      skip 1 -
      center 'Industria'   skip 4;

btitle left 'portfoli.sql';
```

```
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 |
|-------|----------|---------------|---------------|-------|--------------------|------------|
| <hr/> | | | | | | |
| 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 T0 | 77.00 | 76.50 | -.50 | -.65 | 25,205,667 |
| <hr/> | | | | | | |
| | sum | | | | | 39,207,182 |
| <hr/> | | | | | | |
| 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 |
| <hr/> | | | | | | |
| | sum | | | | | 33,615,458 |
| <hr/> | | | | | | |
| 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 |
| <hr/> | | | | | | |
| | sum | | | | | 66,877,300 |

Figura 13.2 Report delle azioni azione con break on Industria e compute sum of Volume.

Ordine delle colonne in break on

Se l'ordine delle colonne in break on non è corretto, possono verificarsi dei problemi. Si supponga di volere un report delle entrate di un'azienda, per Divisione, Dipartimento e Progetto (dove una divisione contiene dipartimenti e i dipartimenti contengono i progetti).

```

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
/

```

| Portfolio corrente | | | 1 April 1998 | | | |
|--------------------|----------|---------------|---------------|-------|--------------------|------------|
| Ind | SOCIETA | Industria | | | | VOLUME |
| | | Chius Ieri | Chius Oggi | NETT | Percento Cambio | |
| 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 13.3 break on e compute sum on Report per totali generali.

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 al cambiare del progetto e inoltre verrebbero accumulati solo per il progetto. Tutto ciò non servirebbe a nulla. Invece, break on deve 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
```

Comandi di formattazione ttitle e btitle

I risultati di questi comandi ttitle e btitle sono visibili nella Figura 13.5, riportata più avanti in questo capitolo.

```
ttitle left 'Portfolio 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 di solito sono variabili; se sono state definite da accept, NEW_VALORE 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 formato.

Per altre opzioni di questi comandi, si può consultare la voce ttitle nella guida alfabetica di riferimento del Capitolo 37.

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.

Aggiunta di viste

Perché il report completo sia utile, è importante effettuare i calcoli di ogni azione in relazione al suo settore industriale e al complesso. Perciò vengono create due viste:

la prima riassume il volume delle azioni raggruppate per Industria, la seconda 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')
/

```

Occorre notare che la pratica di creare viste in un report di SQLPLUS è adatta solo per viste temporanee che non hanno particolare utilizzo al di fuori del report. Viste utilizzate più diffusamente, che vengono condivise da altri e impiegate come se fossero normali tabelle nel database, non dovrebbero essere rimosse né create in un file di avvio SQLPLUS che viene utilizzato solo per un report, ma dovrebbero semplicemente far parte della clausola *from* dell'istruzione *select*.

Colonne utilizzate con **ttitle** e **btitle**

Ora vengono aggiunte alcune definizioni di colonne, insieme a una nuova colonna, PartDiInd, spiegata tra poco (si esamini il punto A nella Figura 13.4).

A questo punto vengono inserite con il comando **NEW_VALORE** due colonne che compariranno in **ttitle** e **btitle**. Si possono vedere le definizioni nel punto B della Figura 13.4 e gli effetti nel punto 1 della Figura 13.5. La colonna Oggi è stata utilizzata per riportare la data di oggi in **btitle**. Come funziona? Innanzitutto Oggi è un alias della colonna **SysDate** formattata nell'istruzione **select**:

```
T0_CHAR(SysDate,'fmMonth ddth, yyyy') Oggi
```

NEW_VALORE pone il contenuto della colonna Oggi (1 aprile 1998) 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 rieseguirlo, possa essere trovato facilmente.

Si noti anche **format 999 sql.pno**. L'ultima parte, **sql.pno**, è una variabile che contiene il numero della pagina corrente ed è possibile utilizzarla ovunque sia in **ttitle** che in **btitle**. Ponendola in **btitle** dopo la parola "right", viene mostrata nell'angolo in fondo 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 al 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----- C
break on Report pag on Industria pag ----- D
compute sum of Volume on Report Industria----- E
compute sum of PartDiInd on Industria----- E
compute avg of Nett Percento on Industria----- E
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 13.4 Comandi SQLPLUS per un report per industria.

format 999, che precede questa variabile, designa 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 in seguito un altro comando di formato.

Un avvertimento riguardo alle variabili Vi sono altri due elementi importanti nel comando column:

```
column Oggi NEW_VALORE xOggi noprint format al trunc
```

| Portfolio 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 1 April 1998 | | | | | | | 1 | Pag 1 |

| Portfolio 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 1 April 1998 | | | | | | | | Pag 2 |

Figura 13.5 Report per industria, con un'industria per pagina. (*continua*)

`noprint` comunica a SQL di non visualizzare questa colonna nella stampa del risultato 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 larghezza di default di circa 100 caratteri (come è stato spiegato nel Capitolo 8). Anche se è attiva l'opzione `noprint`, SQLPLUS conta comunque la larghezza della colonna `Oggi` per stabilire se è stata superata `linesize`. La conseguenza è che viene completamente distrutta la formattazione delle righe in

| Portfolio corrente | | | | | | | SPAZIO |
|------------------------------|----------|---------------|---------------|------|-----------------|--------------------|------------|
| 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 |
| ***** | | | | | | | |
| avg | | | | 1.46 | | 5.30 | |
| sum | | | | | 2.08 | | 66,877,300 |
| portfoli.sql on 1 April 1998 | | | | | | | Pag 3 |

| Portfolio corrente | | | | | | | SPAZIO |
|------------------------------|---------|---------------|---------------|------|-----------------|--------------------|-------------|
| Industria | | | | | | | |
| Ind | SOCIETA | Chius Ieri | Chius Oggi | NETT | Parte di Ind | Percento Cambio | VOLUME |
| (6) | | | | .88 | .19 | 2.66 | 139,699,940 |
| portfoli.sql on 1 April 1998 | | | | | | | Pag 4 |

Figura 13.5 Report per industria, con un'industria per pagina.

visualizzazione e stampa. Cambiando il formato in a1 trunc, questo effetto viene ridotto.

L'altra colonna con NEW_VALORE è Industria. Questa in effetti ha già alcune definizioni di colonna:

```
column Industria 'Ind' format a5 trunc
```

e ora ne viene aggiunta una nuova (si ricordi che istruzioni diverse per una colonna possono essere fornite in righe differenti):

```
column Industria NEW_VALORE xIndustria
```

Come per Oggi, questo comando column inserisce i contenuti della colonna dall'istruzione select in una variabile chiamata Industria. Il valore di questa variabile è coordinato con break on. Essa riceve 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 13.5.

Di seguito sono descritte le azioni intraprese da SQLPLUS.

- 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 restituite tante righe sufficienti a riempire la pagina.
- Stampa ttitle con il valore di xIndustria prima che sia stato cambiato (o prima che la pagina sia diventata piena).
- Sposta il valore di xIndustria in un variabile “OLD_VALORE” e lo salva.
- Stampa le righe sulla pagina.
- Carica il nuovo valore in xIndustria.
- Stampa btitle.
- 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 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, si dovrebbe impiegare la seguente istruzione:

```
column Industria OLD_VALORE xIndustria;
```

Ulteriori informazioni su break on e compute

Il punto C nella Figura 13.4 precede immediatamente i comandi break on e compute. Quando entra in azione un comando compute, 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, pur se viene completamente sostituito da uno nuovo eventualmente presente. È buona pratica inserire comandi clear breaks e clear computes immediatamente prima di impostare nuovi comandi break on e compute.

Ecco le opzioni disponibili per break on:

- break on *colonna*
- break on row
- break on page
- break on report

colonna può essere un nome di colonna, un alias o un'espressione come SUBSTR(Industria,1,4). Ognuna di queste può essere seguita da una delle seguenti azioni:

- 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 page produce un'interruzione ogni volta che viene riempita una pagina (nell'esempio corrente, con questa opzione xIndustria conterrà sempre il valore della colonna Industria per la prima riga della pagina). break on report intraprende l'azione specificata ogni volta che un report viene terminato.

Le azioni skip servono per saltare una o più righe (ovvero stampare delle righe vuote) o per passare all'inizio di una nuova pagina. Si ricordi (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 13.4).

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 noti 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 di AVG(), COUNT(), MAX(), MIN(), STDDEV(), SUM() e VARIANCE() in SQL. Tutti, eccetto num, ignorano NULL e nessuno può utilizzare la parola chiave DISTINCT. compute num è simile a compute count, eccetto che il secondo conta solo le righe non nulle e il primo tutte le righe.

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 di diversi compute contemporaneamente La Figura 13.4 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 13.5, si vedono i loro effetti. Si noti che il calcolo avg per ogni colonna appare su una riga e sum (dove è presente) appare nella riga seguente. Inoltre le parole sum e avg compaiono sotto la colonna che segue la parola on nell'istruzione compute. Mancano al punto 6 per il calcolo dei totali generali perché sono effettuati su Report, che non è una colonna, per cui sum e avg non possono comparire sotto una colonna.

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 noti 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:

$$(ChiusOggi - ChiusIeri)*(S.Volume/I.Volume) \text{ PartDiInd},$$

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) attuale di BRANDON e quella di NORTHERN nell'industria SPAZIO. BRANDON è variata di 0.75, ma con una quantità trattata di 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 di NORTHERN; questo è visibile nei valori relativi per le due aziende sotto la colonna PartDiInd.

Ora si confronti la somma (sum) della colonna PartDiInd, 2.80, 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 il suo valore è ponderato 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 anche per illustrare come vengono calcolate informazioni di riepilogo.

Non è questo il luogo per intraprendere una dettagliata trattazione di medie ponderate, percentuali o calcoli statistici, ma 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 effettua le medie di Nett, Percento e PartDiInd su Report. Questi valori sono mostrati al punto 6 come 0.88, 0.19 e 2.66. Si osservi ora il punto 7 in fondo alla Figura 13.6. Questo è lo stesso report della Figura 13.5, con alcune eccezioni; occupa una pagina invece che di quattro, nel titolo superiore a destra si trova la data, non il settore industriale e la riga in fondo ha un risultato aggiuntivo 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 vengono calcolati dalle colonne visualizzate utilizzando qualsiasi istruzione compute. Questo perché nelle istruzioni compute manca la ponderazione del volume totale dell'industria.

| Portfolio 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 | -.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 |
| | | | | | | | |
| 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 |
| | | | | | | | |
| 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 |
| ***** | | | | | | | |
| avg | | | | 1.46 | | 5.30 | |
| sum | | | | | 2.08 | | 66,877,300 |
| | | | | | | | |
| | | | | .88 | .19 | 2.66 | |
| | | | | | | | 139,699,940 |
| Medie e totali di mercato | | | | 1.09 | 2.66 | 139,699,940 | |

Figura 13.6 Report rivisto, con le medie e i totali corretti del mercato azionario.

Perciò, come è stato prodotto questo risultato? La risposta è fornita nella Figura 13.7; al punto F `btitle` è stato disattivato e al punto G (subito dopo l'istruzione `select` che ha prodotto il corpo principale del report) vi è una `select` 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 `set heading off` è utilizzato per disattivare i titoli delle colonne, che

```

ttitle left 'Portfolio 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 di mercato ', _____ G
       SUM((ChiusOggi-ChiusIeri)*S.Volume)/M.Volume) Nett,
       '',
       AVG((ChiusOggi/ChiusIeri))*100 - 100 Percento,
       SUM(S.Volume) Volume
  from AZIONE S, MERCATO M
 where Industria in ('ELETTRONICA', 'SPAZIO', 'MEDICALE')

```

Figura 13.7 Comandi per la produzione di medie e totali corretti del mercato azionario.

normalmente a questo punto verrebbero visualizzati. Si è dovuto 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 è mostrato nell'output dei report di esempio in questo capitolo (Figure 13.3, 13.5, 13.6), quando viene utilizzato un comando compute, al valore calcolato

viene assegnata un'etichetta con la funzione che è stata eseguita. Ad esempio, nella Figura 13.6, 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.

A partire da ORACLE7.1 è possibile ridefinire le etichette di default e specificare etichette personalizzate per le funzioni calcolate nel comando compute. Per maggiori informazioni si può consultare la clausola label di COMPUTE nella guida alfabetica di riferimento (Capitolo 37).

13.2 set termout off e set termout on

Un'altra utile coppia di comandi è quella costituita da set termout off e set termout on. Il primo è spesso utilizzato in un file di avvio prima del comando spool e il secondo dopo di esso. L'effetto è quello di eliminare la visualizzazione del report sullo schermo. Nella stampa dei report questo fa risparmiare tempo ed evita l'irritante scorrere dei dati sullo schermo. La registrazione su un file continua a funzionare correttamente.

13.3 Variabili in SQLPLUS

Se si utilizza SQLPLUS in modo interattivo, si possono controllare i valori di ttitle e btitle correnti in qualsiasi momento digitando i loro nomi da soli su una riga. SQLPLUS mostra immediatamente il loro contenuto:

```
ttitle
ttitle ON and is the following 90 characters:
left 'Portfolio corrente'           right xOGGI
skip 1      center 'Industria'     ' skip 4
```

Si noti la presenza di xOGGI, che è una variabile, al posto del suo contenuto corrente, come "1 April 1998". SQLPLUS è in grado di salvare e utilizzare molte variabili, quelle impiegate in ttitle e btitle sono soltanto alcune. Le variabili correnti e i loro valori possono essere visualizzati con define:

```
define
DEFINE _SQLPLUS_RELEASE = "400030000" (CHAR)
DEFINE _EDITOR          = "vi" (CHAR)
DEFINE _O_VERSION        = "Oracle8 Server Release 8.0.3.0.0
With the distributed, heterogeneous, replication, objects,
parallel query and Spatial Data options
PL/SQL Release 3.0.3.0.0 " (CHAR)
DEFINE _O_RELEASE         = "800030000" (CHAR)
DEFINE _XINDUSTRIA       = "SPAZIO" (CHAR)
DEFINE XOGGI              = "1 April 1998" (CHAR)
```

La variabile EDITOR indica l'editor che viene utilizzato quando si digita edit. Nel Capitolo 5 si è mostrato come impostarla nel file login.sql. Le altre variabili identificano la versione di ORACLE in uso.

Le ultime variabili sono state definite nelle query del mercato azionario e i loro contenuti sono proprio quelli visualizzati nei report. SQLPLUS salva le variabili per `title` e `btitle` impostandole a un certo valore e poi consente di utilizzarle ogniqualvolta viene eseguita una query. In realtà 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, azione per azione. Questo può essere facilmente ottenuto con delle variabili nella clausola `where`. Normalmente si scriverebbe questa query:

```
select Societa, ChiusIeri, ChiusOggi, Volume
  from AZIONE where Societa = 'IDK';
```

| SOCIETA | Chius | Chius | VOLUME |
|---------|-------|-------|-----------|
| | Ieri | Oggi | |
| IDK | 95.00 | 95.25 | 9,443,523 |

In alternativa la si potrebbe scrivere in un file chiamato, ad esempio, `chiusura.sql`:

```
column ChiusOggi heading 'Chius|Oggi' format 999.90
column ChiusIeri heading 'Chius|Ieri' format 999.90
column Volume format 999,999,999
```

```
accept xSocieta prompt 'Inserire nome Societa: '

select Societa, ChiusIeri, ChiusOggi, Volume
  from AZIONE
 where Societa = '&xSocieta';
```

In questo file `xSocieta` è una variabile inventata, come `xIndustria` o `xOggi`. `accept` comunica a SQLPLUS di accettare input dalla tastiera e `prompt` di visualizzare un messaggio. Si noti che il nome della variabile, quando si trova nell'istruzione SQL `select`, deve 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:

Si può digitare **MEMORY GRAPHICS** e 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'
```

| SOCIETA | Chius | Chius | VOLUME |
|-----------------|-------|-------|-----------|
| | Ieri | Oggi | |
| MEMORY GRAPHICS | 15.50 | 14.25 | 4,557,992 |

Innanzitutto viene mostrata la query che è stata costruita, poi 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 dell'azienda, i valori old e new mostrati sono i seguenti:

```
old  3: where Societa = 'MEMORY GRAPHICS'
new  3: where Societa = 'IDK'
```

e il risultato riguarda IDK. Non è necessario che 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 questi comandi sono impostati attualmente, occorre digitare il comando show seguito dalla parola chiave verify o echo.

```
show verify
verify ON
```

```
show echo
echo ON
```

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

set echo off
set verify off
set sqlcase upper
accept xSocieta prompt 'Inserire nome Societa: '

select Societa, ChiusIeri, ChiusOggi, Volume
  from STOCK
 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 si sono salvati i dati in lettere maiuscole e non si vuole indurre le persone a digitare necessariamente in maiuscolo, quando desiderano eseguire una query. Ora, quando il file di avvio viene eseguito, accade quanto segue:

```
start closing.sql
```

```
Inserire nome Societa: memory graphics
```

| SOCIETA | Chius Ieri | Chius Oggi | VOLUME |
|-----------------|---------------|---------------|-----------|
| MEMORY GRAPHICS | 15.50 | 14.25 | 4,557,992 |

L'utilizzo di variabili nei file di avvio può essere molto utile, in particolare per report eseguiti su una base sviluppata ad hoc, dove il formato fondamentale rimane invariato ma determinati parametri cambiano, come data, divisione dell'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 finché non viene rimossa con undefined:

```
undefined xSocieta
```

Se ora si cerca di vedere il valore della variabile si ottiene quanto segue:

```
define xSocieta
```

```
symbol xSocieta is UNDEFINED
```

Si può anche definire una variabile nel file di avvio senza impiegare il comando accept, ma assegnandole semplicemente un valore, come mostrato qui:

```
define xSocieta = 'IDK'
```

In ogni punto del file di avvio in cui compare, xSocieta viene sostituita da 'IDK'.

Altri luoghi dove utilizzare le variabili

Qualsiasi variabile definita utilizzando accept o define può essere impiegata direttamente in btitle o ttitle senza utilizzare il comando di colonna NEW_VALORE. Tutto quello che fa NEW_VALORE è prendere il contenuto di una colonna ed emettere il proprio comando define per la variabile che 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 anche essere impiegate in un file di avvio di configurazione, come viene spiegato più avanti in questo capitolo nel paragrafo “Utilizzo di mask.sql”.

13.4 Formattazione di numeri

Il metodo di default utilizzato da SQLPLUS per formattare i numeri consiste semplicemente nell'allinearli a destra in una colonna, senza virgole di separazione delle migliaia, utilizzando il punto decimale solo se il numero non è un intero. La tabella TESTONUMERO contiene le colonne Valore1 e Valore2, che hanno numeri identici e vengono utilizzate per mostrare come funziona la formattazione di numeri.

La prima query mostra la formattazione di default:

```
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 |
| 98761234.567 | 98761234.567 |

La quinta riga è NULL. Si noti 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 di default 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 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 |

Lasciando questo formato di default, la formattazione di Valore1 può essere cambiata. Lo zero alla fine assicura che in ogni riga con un valore venga prodotto

almeno uno zero. Poiché in entrambi i formati non sono state specificate cifre decimali, .0001 e 0 (nelle prime due righe) vengono visualizzati come zero.

```
column Valore1 format 99,999,990
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 |

Altre opzioni per la formattazione sono descritte nel paragrafo seguente.

Opzioni per la formattazione di numeri

Le seguenti opzioni funzionano con i comandi set numformat e column format.

- 9999990 Il numero di nove e zero totali determina il numero massimo di cifre che possono essere visualizzate.
- 999,999,999.99 Nel modello mostrato vengono inserite virgole di separazione delle migliaia e punti decimali. Non viene visualizzato niente se il valore è zero.
- 999990 Visualizza uno zero se il valore è zero.
- 099999 Visualizza numeri con zero iniziali.
- \$99999 Davanti a ogni numero viene posto il segno del dollaro.
- B99999 Non viene visualizzato nulla se il valore è zero. Questa è la formattazione di default.
- 99999MI Se il numero è negativo, il segno meno segue il numero. Per default il segno negativo è a sinistra.
- 99999PR I numeri negativi vengono visualizzati con < e >.
- 9.999EEEE La visualizzazione è in notazione scientifica.
- 999V99 Moltiplica i numeri per 10ⁿ, dove *n* indica il numero di cifre a destra di V. 999V99 trasforma 1234 in 123400.

In alternativa, questa riga potrebbe essere inserita nel file login.sql descritto nel Capitolo 5; in questo caso viene caricata automaticamente ogni volta che viene avviato SQLPLUS. Si noti la riga finale del file mask.sql. Qui possono anche essere definite delle variabili, che potrebbero essere utilizzate per impostare informazioni specifiche della posizione da impiegare in clausole title, btitle o where per i report.

13.5 Utilizzo 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 i comandi fondamentali in un singolo file. Questo file potrebbe essere chiamato *mask.sql*, perché contiene le informazioni di formattazione (o “maschera”) per le colonne. Il file potrebbe essere, ad esempio, come quello 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 poi viene incorporato in un file di avvio (di solito all'inizio) semplicemente inserendo questa riga:

```
start mask.sql
```

13.6 Utilizzo di buffer per salvare comandi SQLPLUS

Normalmente l'utilizzo dell'editor della riga di comando in SQLPLUS (descritto nel Capitolo 5) consente di agire solo sull'istruzione SQL. È per questo motivo che in genere viene utilizzato un editor differente per modificare i file di avvio. Vi è un'altra alternativa, adatta solo all'inizio dello sviluppo di file di avvio, poiché può generare confusione se impiegata con file più grandi.

Quando viene utilizzato l'editor della riga di comando, i comandi SQL (ma non quelli di SQLPLUS) vengono salvati in un'area temporanea della memoria del computer chiamata *buffer SQL*. Quando si modificano le righe, utilizzando change, append, del e così via, in realtà cambiano le righe salvate nel buffer SQL. Poiché quando si inizia a lavorare su una query si desiderano cambiare non solo i comandi SQL ma anche quelli SQLPLUS come column, ttitle e simili, ORACLE consente di utilizzare un buffer per salvarli entrambi.

Il buffer di default è chiamato *sql*; se con il comando set buffer gli si assegna un altro nome (lungo fino a sette caratteri), in questo buffer vengono salvati anche i comandi SQLPLUS. Ad esempio, per creare un buffer chiamato *A* si può iniziare inserendovi input, poi una definizione column e un'istruzione select. Ecco ciò che si dovrebbe digitare:

```
set buffer A
input
```

```
column Volume format 999,999,999
select Societa, Volume from AZIONE;
```

set buffer A riserva l'area di memoria temporanea per quello che segue. input comunica a SQLPLUS che si sta iniziando a immettere informazioni in questa area. Vengono inserite due righe, seguite da una riga vuota (ottenuta con INVIO) per comunicare a SQLPLUS che si è terminato di digitare le informazioni. Il buffer è ora “ pieno ” e chiuso. Questo significa, ovviamente, che non è possibile utilizzare questo metodo per inserire righe vuote in un file. Si può impiegare show per controllare se il buffer è ancora attivo (ovvero se è il buffer attualmente in uso) come mostrato di seguito:

```
show buffer
```

```
buffer A
```

Se è così, si possono utilizzare i comandi list, del, append e change dell'editor della riga di comando su qualsiasi riga in esso contenuta (si rimanda al Capitolo 5).

```
list
```

```
1 column Volume format 999,999,999
2* select Societa, Volume from AZIONE;
```

Si può anche salvare (save) il contenuto del buffer in un file e riportarlo (get) dal file nel buffer per ulteriori modifiche. In questo esempio le righe precedenti sono salvate in un file chiamato check:

```
save check
```

```
Wrote file check
```

L'opposto di save è get. Se si ha un file di avvio da modificare con l'editor della riga di comando, lo si può ad esempio spostare in un buffer chiamato test digitando:

```
set buffer test
```

```
get control
```

In questo modo tutto quello che c'è in check viene spostato nel buffer test, dove può essere modificato dall'editor della riga di comando. Sfortunatamente non è possibile eseguire comandi SQLPLUS o SQL nel buffer test. Si possono salvare i cambiamenti apportati utilizzando di nuovo il comando save check (con il parametro repl per sostituire il file check.sql corrente) e si può eseguire il file digitando **start check**, ma non vi è alcun modo per eseguire direttamente il buffer. Si può tuttavia eseguire il buffer SQL, anche se esistono alcune restrizioni.

- Se il buffer è sql, qualsiasi istruzione SQL digitata direttamente sulla riga di comando in SQLPLUS va nel buffer SQL. Se viene digitato un punto e virgola (;) al termine di una parola, l'istruzione SQL nel buffer viene eseguita immediatamente.
- Dopo la sua esecuzione, nel buffer sql rimane solo l'istruzione, senza il punto e virgola. Per rieseguirla occorre digitare una barra (/) o la parola run da sola su una riga, comunicando così a SQLPLUS di eseguire tutto ciò che vi è nel buffer sql.

- Se il buffer corrente è sql, get vi trasferisce il contenuto di un file. Però, se il file contiene qualsiasi comando SQLPLUS o un punto e virgola, non è possibile eseguirlo.
- Ogniqualvolta si esegue un file di avvio, le istruzioni SQL in esso presenti vengono automaticamente caricate nel buffer sql e vengono rieseguite se si digita run o la barra /.
- Ogni volta che viene eseguito un comando SQL, dalla riga di comando o da un file di avvio, il buffer viene automaticamente impostato in sql. Il contenuto del buffer A non scompare, ma per modificarlo occorre digitare di nuovo set buffer A.

Di seguito sono descritte sono le regole fondamentali per modificare un buffer che non sia il buffer SQL.

- set buffer deve essere seguito da una parola lunga da uno a sette caratteri.
- Deve essere digitata la parola input da sola su una riga.
- Devono essere digitati i comandi SQLPLUS, compresi titoli, colonne, e simili.
- Deve essere digitata l'istruzione SQL.
- Deve essere inserita una riga vuota.
- Il buffer deve essere salvato in un file.

Si può modificare il contenuto del buffer prima che venga salvato nel file utilizzando l'editor della riga di comando. Se il file viene richiamato con il comando get, il suo contenuto viene trasferito nel buffer corrente (impostato con l'ultimo set). In una singola sessione SQLPLUS si possono avere diversi buffer, ognuno con contenuti differenti. Tutti i buffer vengono cancellati quando si abbandona SQLPLUS. Se si desidera conservare il lavoro svolto su un buffer, occorre assicurarsi di salvarlo in un file prima di uscire da SQLPLUS.

La situazione sembra abbastanza problematica e in effetti può esserlo. Le regole sono difficili da ricordare; distinguere quale buffer è attivo e se è possibile eseguirlo, o (nel caso di un buffer sql) se il contenuto è eseguibile, può mettere in difficoltà anche un utente ORACLE esperto. I buffer e l'editor della riga di comando vanno bene solo per modifiche semplici e immediate e non come un completo ambiente di sviluppo. Generalmente è molto più facile utilizzare semplicemente il proprio editor, includervi i comandi SQLPLUS e SQL e avviarlo in SQLPLUS.

Registrazione delle definizioni correnti

Esiste un altro metodo per salvare i comandi SQLPLUS. La maggior parte dei comandi può essere visualizzata utilizzando il comando show o le parole column, define, ttitle o btitle da sole. Se si desidera inserire tutti gli output in un file, occorre semplicemente utilizzare il comando spool, come mostrato di seguito:

```
spool saveit.sql
```

Il comando spool salva tutto quello che appare sullo schermo in un file, ad esempio il file saveit.sql. Una volta che spool è attivo, si provi a digitare quanto segue:

```
column
ttitle
btitle

spool off
```

In questo modo tutte le definizioni column, ttitle e btitle correnti vengono caricate nel file saveit.sql, che può essere poi modificato. I comandi column non sono riportati precisamente nell'ordine corretto, ma risultano abbastanza vicini alla forma giusta e possono essere modificati rapidamente. Lo stesso vale per i titoli.

A partire da ORACLE7.3 si può utilizzare il comando store per salvare tutte le impostazioni dell'ambiente SQLPLUS in un file. Un esempio del comando store è mostrato nel seguente listato:

```
store setting.sql
```

In questo modo viene creato il file SETTING.SQL, che conterrà tutti i comandi ttitle, btitle e set, i quali potranno essere eseguiti con il comando start.

13.7 **show all e spool**

Sono stati trattati vari comandi che utilizzano il comando set e di cui è possibile determinare lo stato corrente con il comando show, come feedback, echo, verify, heading e così via. Vi sono in realtà circa 50 comandi che possono essere impostati ed è possibile visualizzarli tutti con:

```
show all
```

Sfortunatamente i comandi scorrono così velocemente sullo schermo che risulta impossibile leggerli. Si può risolvere questo problema registrandoli. Occorre semplicemente 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 esaminare lo stato corrente di tutti i comandi set richiamando il file dal proprio editor. È possibile trovare questi comandi nella guida alfabetica di riferimento riportata nel Capitolo 37, sotto la voce set. A partire da ORACLE7.3 le righe restituite dal comando show all sono in ordine alfabetico.

13.8 **Aggiunta di righe vuote**

Le informazioni restituite da un database spesso vanno bene così come sono, con una riga con i titoli di colonna e le colonne dei dati disposte sotto di essa. In alcune occasioni, però, è preferibile una disposizione diversa, che a volte può essere realizzata utilizzando colonne letterali con spazi vuoti, per poter posizionare in modo appropriato i dati reali su più di una riga 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 riga dei totali nella Figura 13.6. Ad 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
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 | 0 | 18,333,876 |
| PUBBLICITA | | | MBK COMMUNICATIONS |
| 43.25 | 41 | -2.25 | 10,022,980 |
| PUBBLICITA | | | NANCY LEE FEATURES |
| 13.5 | 14.25 | .75 | 14,222,692 |
| VESTIARIO | | | ROBERT JAMES APPAREL |
| 23.25 | 24 | .75 | 19,032,481 |

Con il comando column è stata definita, con un alias B, una colonna letterale di uno spazio, larga 21 caratteri e con un'intestazione vuota. Essa viene utilizzata per spostare i nomi delle aziende in modo che le colonne siano allineate come si desidera, con il volume delle azioni direttamente sotto il nome dell'azienda e ChiusOggi e Nett allineate con gli spazi vuoti.

fold_after e fold_before

È opportuno esaminare come il comando fold_after agisce su ogni colonna nell'istruzione select:

```
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 risultato:

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

`fold_after` opera dopo ogni colonna, eccetto `ChiusOggi` perché il comando `fold_after` a essa applicato non è seguito da un numero. Questo numero può essere un intero qualsiasi, anche se si consiglia il valore 1. Il numero in realtà non serve assolutamente a nulla (deve solo essere presente), ma potrebbe essere utilizzato in una futura versione di ORACLE per indicare il numero di righe di cui occorre avanzare. `fold_before` produce un risultato simile, ma inserisce una riga vuota prima della stampa di una colonna, invece che dopo.

Un utilizzo naturale di `fold_after` e `fold_before` si ha nella preparazione di etichette per la posta, elenchi di clienti e così via.

13.9 Ulteriori controlli per la realizzazione di report

Molti dei comandi illustrati in questo come in altri capitoli hanno diverse altre opzioni oltre a quelle utilizzate in questi esempi. Tutte le opzioni per ciascuno dei comandi possono essere trovate nella guida alfabetica di riferimento del Capitolo 37, sotto il nome dei singoli comandi.

Capitolo 14

Modifica dei dati: insert, update e delete

14.1 **insert**

14.2 **rollback, commit e autocommit**

14.3 **delete**

14.4 **update**

Tutto ciò che si è appreso finora su ORACLE, SQL e SQLPLUS riguarda la selezione di dati da tabelle nel database. Questo capitolo mostra come modificare i dati in una tabella: come inserire (insert) nuove righe, come aggiornare (update) i valori delle colonne nelle righe e come cancellare (delete) completamente delle righe.

Anche se questi argomenti non sono stati trattati esplicitamente, quasi tutto il materiale appreso circa SQL, compresi tipi di dati, calcoli, formattazione di stringhe, clausole where e simili, può essere utilizzato qui, perciò non vi è molto altro da imparare. ORACLE fornisce anche una funzionalità per inserire, aggiornare e cancellare dati in database remoti (trattata nel Capitolo 21).

14.1 **insert**

Il comando SQL insert consente di inserire una riga di informazioni direttamente in una tabella (o indirettamente attraverso una vista; per questo si rimanda al paragrafo “Creazione di una vista”). La tabella COMFORT registra le temperature a mezzogiorno e a mezzanotte e le precipitazioni giornaliere, città per città, per una serie di quattro date campione nel corso dell’anno:

```
describe COMFORT
```

| Name | Null? | Type |
|----------------|-------|--------------|
| CITTA | | VARCHAR2(13) |
| DATACAMPIONE | | DATE |
| MEZZOGIORNO | | NUMBER |
| MEZZANOTTE | | NUMBER |
| PRECIPITAZIONE | | NUMBER |

Per aggiungere una nuova riga a questa tabella, occorre utilizzare la seguente istruzione:

```
insert into COMFORT
values ('WALPOLE', TO_DATE('21-MAR-1993','DD-MON-YYYY'),
       13.8, 6.6, 0);
```

1 row created.

La parola values deve precedere l'elenco di dati da inserire. Si noti che una stringa di caratteri deve essere racchiusa fra apici singoli, mentre i numeri possono stare da soli. I campi devono essere separati da virgole e devono avere lo stesso ordine delle colonne nella descrizione della tabella.

Una data deve essere espressa nel formato di default di ORACLE, racchiusa fra apici singoli. 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('06/22/1993','MM/DD/YYYY'),
       13.8, 6.6, 0);
```

1 row created.

Inserimento di un'ora

Entrambi questi metodi di inserimento delle date producono un'ora di default corrispondente alla mezzanotte, l'inizio del giorno. Se si desidera inserire una data con un'ora differente, occorre semplicemente impiegare la funzione TO_DATE e includere un'ora:

```
insert into COMFORT
values ('WALPOLE', TO_DATE('06/22/1993 1:35',
                           'MM/DD/YYYY HH24:MI'), 13.8, 6.6, 0);
```

1 row created.

Le colonne possono essere inserite 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, ma consente semplicemente di elencare i campi dei dati in un ordine diverso.

NOTA *Si può anche inserire un valore NULL, il quale indica semplicemente che la colonna in questa riga deve rimanere vuota, come viene mostrato nel listato seguente:*

```
insert into COMFORT
(DataCampione, Precipitazione, Citta, Mezzogiorno, Mezzanotte)
values (TO_DATE('23-SEP-1993','DD-MON-YYYY'), NULL,
       'WALPOLE', 30.2, 22.3);
```

1 row created.

NOTA Per informazioni sull'inserimento di dati negli oggetti in ORACLE8 si devono consultare i Capitoli 25 e 26.

insert con select

È anche possibile inserire informazioni che sono state selezionate da una tabella. Di seguito vengono inserite delle colonne selezionate dalla tabella COMFORT, insieme a valori letterali per DataCampione (22-DEC-93) e Citta (WALPOLE). Si noti la clausola where nell'istruzione select, che recupera solo una riga. Se con select si fossero ottenute 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 TO_DATE('22-DEC-1993','DD-MON-YYYY'), Precipitazione,
      'WALPOLE', Mezzogiorno, Mezzanotte
    from COMFORT
   where Citta = 'KEENE' and DataCampione = '22-DEC-93';
```

1 row created.

NOTA Non è possibile utilizzare la sintassi insert into... select from con tipi di dati LONG.

Naturalmente non è necessario inserire il valore della colonna selezionata, ma è anche possibile modificare la colonna impiegando nell'istruzione select qualsiasi funzione stringa di data o numerica appropriata. In questo caso vengono inseriti i risultati di queste funzioni. Si può provare a inserire un valore che supera la larghezza (per i caratteri) o la precisione (per i numeri) della colonna. Però, poiché 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 "mismached 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-93 | 13.7 | 6.6 | 0 |
| WALPOLE | 22-JUN-93 | 13.7 | 6.6 | 0 |
| WALPOLE | 23-SEP-93 | 30.2 | 22.3 | |
| WALPOLE | 22-DEC-93 | -21.8 | -18.4 | 3.9 |

4 rows selected.

A partire da ORACLE8 si può utilizzare APPEND per migliorare le prestazioni di inserimenti lunghi. Così si comunica al database di trovare l'ultimo blocco nel quale i dati della tabella sono stati inseriti. I nuovi record vengono inseriti a partire dal blocco successivo, subito dopo quello impiegato precedentemente.

Ad esempio, se una tabella ha utilizzato precedentemente 20 blocchi nel database, un comando `insert` che impiega l'hint APPEND inizia a scrivere i dati a cominciare dal ventunesimo blocco. Dal momento 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 *hint* appare come un commento, inizia con `/*` e termina con `*/`. La sola differenza nella sintassi è che la serie iniziale dei caratteri include un '+' prima del nome dell'*hint*. Il seguente esempio mostra un comando `insert` i cui dati vengono aggiunti alla tabella.

```
insert /*+ APPEND */ into LAVORATORE (Nome)
select Nome from PROSPETTIVA;
```

I record della tabella PROSPETTIVA vengono inseriti nella tabella LAVORATORE. Invece di tentare di riutilizzare lo spazio impiegato precedentemente nella tabella LAVORATORE, i nuovi record vengono posti alla fine della spazio di memoria fisico della tabella.

Dal momento che i nuovi record non tentano di riutilizzare lo spazio disponibile, le esigenze di spazio della tabella LAVORATORE potrebbero aumentare. In generale, si dovrebbe utilizzare APPEND quando si inseriscono grandi quantità di dati in tabelle con poco spazio riutilizzabile. Il punto dove vengono aggiunti i record è detto *highwatermark* della tabella e l'unico modo di azzerare un highwatermark è quello di troncare con `truncate` la tabella stessa.

Dal momento 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` nella guida alfabetica di riferimento (Capitolo 37).

14.2 rollback, commit e autocommit

Quando si inseriscono, si aggiornano o si cancellano dati dal database, si può "invertire" o "annullare" l'operazione con il *rollback*. Questo 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, in base alla caratteristica `autocommit` di set. Come per altre caratteristiche, è possibile visualizzare l'impostazione corrente di autocommit in questo modo:

```
show autocommit
```

```
autocommit OFF
```

`OFF` è il valore di default. A partire da ORACLE7.2 si può anche specificare un numero per il valore set `autocommit`, in modo da determinare il numero di comandi dopo i quali ORACLE emette 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, mentre per chi sta effettuando il lavoro le nuove informazioni sono visibili ogni volta che effettua una selezione nella tabella. In effetti si sta operando su un'area “provvisoria”, 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 interpretato male. In realtà significa soltanto che deve essere annullata ogni operazione che non è stata confermata.

Se si conferma una serie di transazioni, esplicitamente con `commit` o implicitamente con un'altra azione (un esempio verrà mostrato in seguito), il messaggio “`rollback complete`” in effetti non significa nulla. I paragrafi “Commit e rollback” e “Commit e rollback in SQL” offrono un quadro del modo in cui `commit` e `rollback` agiscono su una tabella di nomi di cani.

commit隐式

Le azioni che fanno in modo che si verifichi un `commit`, anche senza alcuna istruzione esplicita, sono `quit`, `exit` (equivalente a `quit`), `create table` o `create view`, `drop table` o `drop view`, `grant` o `revoke`, `connect` o `disconnect`, `alter`, `audit` o `noaudit`. Utilizzare uno di questi comandi significa eseguire automaticamente un `commit`.

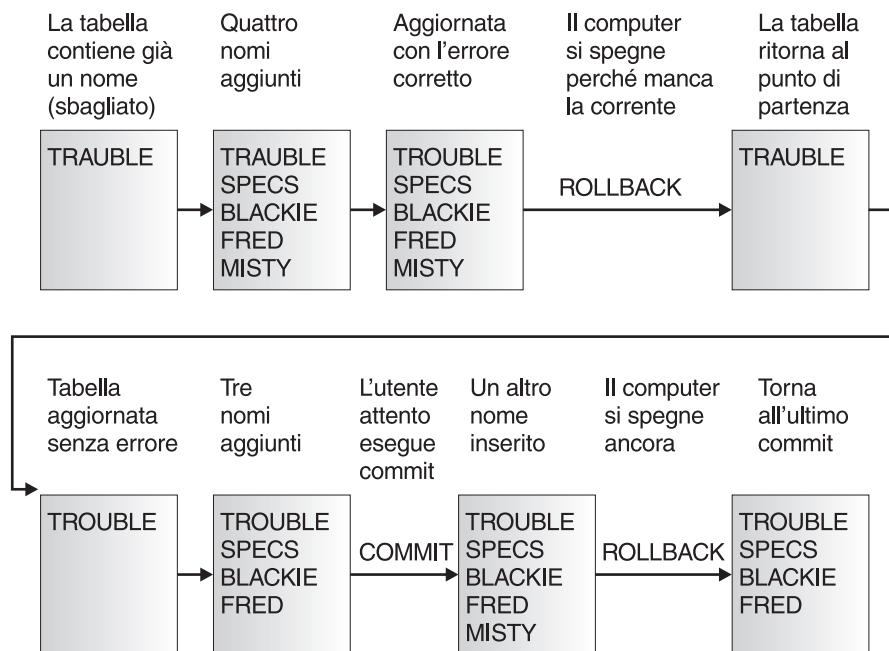
rollback自动

Se si è completata una serie di inserimenti, aggiornamenti o cancellazioni, ma non la si è confermata esplicitamente o implicitamente, e si incorre in serie difficoltà, come un problema al computer, ORACLE annulla automaticamente qualsiasi operazione non confermata.

Se la macchina o il database si blocca, questo lavoro di pulizia viene effettuato la volta successiva che il database viene richiamato.

Commit e rollback

Si supponga di avere una tabella di database con una sola colonna e una sola riga. La tabella è CANE e la colonna Nome. Contiene già un solo cane, TROUBLE, il cui nome è stato scritto in modo errato come "TRAUBLE" e il diagramma mostra i tentativi del padrone di TROUBLE di aggiungere un altro cane alla tabella e di correggere il nome di quello già presente. Si presuppone che la caratteristica autocommit di SQLPLUS sia disattivata. Per l'equivalente SQL delle azioni di questo diagramma occorre consultare il paragrafo seguente.



Misty deve essere aggiunto di nuovo quando l'utente riaccende il computer.

Commit e rollback in SQL

In SQL, la sequenza precedente appare in questo modo:

```
select *from CANE;
```

NOME

TRAUBLE

```
insert into CANE values ('SPECS');
insert into CANE values ('BLACKIE');
insert into CANE values ('FRED');
insert into CANE values ('MISTY');

update CANE set Nome = 'TROUBLE' where Nome = 'TRAUBLE';

rollback;

select *from CANE:

NOME
-----
TRAUBLE

update CANE set Nome = 'TROUBLE' where Nome = 'TRAUBLE';

insert into CANE values ('SPECS');
insert into CANE values ('BLACKIE');
insert into CANE values ('FRED');

commit;

insert into CANE values ('MISTY');

rollback;

select *from CANE:

NOME
-----
TROUBLE
SPECS
BLACKIE
FRED
```

14.3 delete

La rimozione di una o più righe da una tabella richiede il comando `delete`. La clausola `where` è essenziale per eliminare solo le righe desiderate. `delete` senza una clausola `where` svuota completamente la tabella.

```
delete from COMFORT where Citta = 'WALPOLE';
```

4 rows deleted.

Naturalmente una clausola `where` in `delete`, proprio come in un'istruzione `update` o `select` che fa parte di `insert`, può contenere tanta logica di programmazione quanto

qualsiasi istruzione select e può includere sottoquery, unioni, intersezioni e così via. Si può sempre annullare un comando insert, update o delete sbagliato, ma in realtà conviene sempre provare prima 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 selectede

Ora si provi ad annullare delete ed eseguire di nuovo la query:

```
rollback;  
rollback complete
```

```
select * from COMFORT  
where Citta = 'WALPOLE';
```

| CITTA | DATA | CAMPIONE | MEZZOGIORNO | MEZZANOTTE | PRECIPITAZIONE |
|---------|-----------|----------|-------------|------------|----------------|
| WALPOLE | 21-MAR-93 | | 13.7 | 6.6 | 0 |
| WALPOLE | 22-JUN-93 | | 13.7 | 6.6 | 0 |
| WALPOLE | 23-SEP-93 | | 30.2 | 22.3 | |
| WALPOLE | 22-DEC-93 | | -21.8 | -18.4 | 3.9 |

4 rows selected.

Questo illustra che il recupero è possibile, finché non si è 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. Una transazione a cui è stato applicato truncate non può essere annullata o confermata; i record troncati sono irrecuperabili. Per ulteriori dettagli si può consultare la voce truncate nella guida alfabetica di riferimento (Capitolo 37).

14.4 **update**

update richiede di impostare valori particolari per ogni colonna che si desidera modificare e di specificare su quale riga o su quali righe si desidera operare utilizzando una clausola where costruita attentamente:

```
update COMFORT set Precipitazione = .5, Mezzanotte = 22.9  
where Citta = 'WALPOLE'  
and DataCampione = TO_DATE('22-DEC-1993','DD-MON-YYYY');
```

1 row updated.

Ecco l'effetto:

```
select * from COMFORT
where Citta = 'WALPOLE';
```

| CITTA | DATA | CAMPIONE | MEZZOGIORNO | MEZZANOTTE | PRECIPITAZIONE |
|---------|-----------|----------|-------------|------------|----------------|
| WALPOLE | 21-MAR-93 | | 13.7 | 6.6 | 0 |
| WALPOLE | 22-JUN-93 | | 13.7 | 6.6 | 0 |
| WALPOLE | 23-SEP-93 | | 30.2 | 22.3 | |
| WALPOLE | 22-DEC-93 | | -21.8 | 22.9 | .5 |

4 rows selected.

Che si fa se si scopre che il termometro utilizzato in Walpole riporta le temperature costantemente con un grado di meno? Si possono anche svolgere calcoli, funzioni stringa e ogni altra legittima funzione 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 Walpole sono aumentate di un grado:

```
update COMFORT set Mezzanotte=Mezzanotte+1, Mezzogiorno=Mezzogiorno+1
where Citta = 'WALPOLE';
```

4 rows updated.

Ed ecco l'effetto dell'aggiornamento:

```
select * from COMFORT
where Citta = 'WALPOLE';
```

| CITTA | DATA | CAMPIONE | MEZZOGIORNO | MEZZANOTTE | PRECIPITAZIONE |
|---------|-----------|----------|-------------|------------|----------------|
| WALPOLE | 21-MAR-93 | | 14.7 | 7.6 | 0 |
| WALPOLE | 22-JUN-93 | | 14.7 | 7.6 | 0 |
| WALPOLE | 23-SEP-93 | | 31.2 | 23.3 | |
| WALPOLE | 22-DEC-93 | | -20.8 | 23.9 | .5 |

4 rows selected.

Anche qui, come in delete, la clausola where è critica; senza di essa viene aggiornata ogni riga del database. Con una clausola where costruita in modo non corretto, vengono aggiornate le righe sbagliate, spesso in una maniera difficile da scoprire e complessa da risolvere, soprattutto se l'operazione è stata confermata.

Quando si effettuano aggiornamenti conviene sempre utilizzare l'impostazione set feedback on ed esaminare il risultato per assicurarsi che il numero sia quello previsto. È 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 noti che questa select ha la propria clausola where, che preleva la temperatura di MANCHESTER dalla tabella CLIMA, e update ha la propria clausola where che influisce solo sulla città WALPOLE in un certo giorno:

```
update COMFORT set Mezzanotte =
  (select Temperatura
   from CLIMA
   where Citta = 'MANCHESTER')
where Citta = 'WALPOLE'
  AND DataCampione = TO_DATE('22-DEC-1993','DD-MON-YYYY');
```

1 row updated.

Ecco l'effetto dell'aggiornamento:

```
select * from COMFORT
  where City = 'WALPOLE';
```

| CITTA | DATA | CAMPIONE | MEZZOGIORNO | MEZZANOTTE | PRECIPITAZIONE |
|---------|-----------|----------|-------------|------------|----------------|
| WALPOLE | 21-MAR-93 | | 14.7 | 7.6 | 0 |
| WALPOLE | 22-JUN-93 | | 14.7 | 7.6 | 0 |
| WALPOLE | 23-SEP-93 | | 31.2 | 23.3 | |
| WALPOLE | 22-DEC-93 | | -21.2 | 18.9 | .5 |

4 rows selected.

Quando si utilizza una sottoquery con update, occorre assicurarsi che non restituisca più di un record per ognuno dei record aggiornati, altrimenti l'aggiornamento fallisce. Per maggiori dettagli sulle query correlate si può consultare il Capitolo 11.

Si può anche utilizzare un'istruzione select incorporata per aggiornare colonne multiple contemporaneamente. Le colonne devono essere racchiuse fra parentesi e separate da virgolette, come mostrato qui:

```
update COMFORT set (Mezzogiorno, Mezzanotte) =
  (select Umidita, Temperatura
   from CLIMA
   where Citta = 'MANCHESTER')
where Citta = 'WALPOLE'
  AND DataCampione = TO_DATE('22-DEC-1993','DD-MON-YYYY');
```

1 row updated.

Ed ecco il risultato:

```
select * from COMFORT
  where Citta = 'WALPOLE';
```

| CITTA | DATA | CAMPIONE | MEZZOGIORNO | MEZZANOTTE | PRECIPITAZIONE |
|---------|-----------|----------|-------------|------------|----------------|
| WALPOLE | 21-MAR-93 | | 14.7 | 7.6 | 0 |
| WALPOLE | 22-JUN-93 | | 14.7 | 7.6 | 0 |
| WALPOLE | 23-SEP-93 | | 31.2 | 23.3 | |
| WALPOLE | 22-DEC-93 | | 98 | 18.9 | .5 |

4 rows selected.

Si può anche generare una serie di update utilizzando SQLPLUS come un generatore di codice. Questo argomento verrà trattato nel Capitolo 20.

update con NULL

Si può anche aggiornare una tabella e porla uguale a NULL. Questo è il solo caso in cui si può utilizzare con NULL il segno di uguale, invece della parola “is”.

Ad esempio, questo listato:

```
update COMFORT set Mezzogiorno = NULL
where City = 'WALPOLE'
AND DataCampione = TO_DATE('22-DEC-1993','DD-MON-YYYY');
```

1 row updated.

imposta a NULL la temperatura di mezzogiorno per Walpole al 22 dicembre 1993.

NOTA I principali problemi con insert, update e delete consistono nell'attenta costruzione delle clausole where per operare (o inserire) 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 corretto. Questi tre comandi estendono la potenza di ORACLE molto al di là delle semplici query e consentono la manipolazione diretta dei dati.

• Capitolo 15

• **Utilizzo avanzato di funzioni e variabili**

• 15.1 **Funzioni in order by**

• 15.2 **Diagrammi a barre e grafici**

• 15.3 **Utilizzo di TRANSLATE**

• 15.4 **Copia e incollamento complessi**

• 15.5 **Conteggio delle ricorrenze di stringhe in stringhe più lunghe**

• 15.6 **Variabili e sostituzioni registrate**

Nei precedenti capitoli sono stati presentati definizioni ed esempi 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 per essere in grado di costruire combinazioni di funzioni anche abbastanza complesse.

Questo capitolo amplia alcuni degli argomenti esposti in precedenza e mostra 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 utilizzando SQL e gli esempi in questo capitolo possono aiutare a sviluppare le capacità per risolvere anche i problemi reali.

15.1 **Funzioni in order by**

Per cambiare l'ordinamento della sequenza in order by si possono utilizzare anche le funzioni. In questo esempio si impiega la funzione SUBSTR per porre 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 | |

15.2 Diagrammi a barre e grafici

In SQLPLUS è anche possibile produrre semplici diagrammi a barre e grafici, utilizzando un calcolo numerico nella funzione LPAD. Innanzitutto occorre esaminare i comandi di formattazione delle colonne:

```
column Nome format a16
column Eta Format 999
column Grafico Heading 'Eta|    1    2    3    4    5    6    7-
|....0....0....0....0....0' justify c
column Grafico format a35
```

Le prime due righe sono chiare; la terza richiede alcune spiegazioni. Il trattino alla fine di questa riga comunica a SQLPLUS che il comando di colonna continua alla riga successiva. Se alla fine di una riga è presente un trattino, SQLPLUS pre-suppone che seguia un'altra riga dello stesso comando.

ATTENZIONE *SQLPLUS inserisce uno spazio in un'intestazione dove trova un trattino, perciò è importante porre il segno | all'inizio della seconda riga invece che alla fine della prima.*

Se il comando fosse stato simile a questo:

```
column Grafico Heading 'Eta|    1    2    3    4    5    6    7|-|
....0....0....0....0....0' justify c
```

sarebbe stata prodotta la seguente intestazione:

| Eta | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------------------------------------|---|---|---|---|---|---|---|
|0....0....0....0....0....0....0 | | | | | | | |

invece di quella illustrata nella Figura 15.1. Di seguito è riportata l'istruzione SQL che produce il diagramma a barre orizzontali. I lavoratori la cui età è sconosciuta non vengono inclusi.

```
select Nome, Eta, LPAD('o',ROUND(Eta/2,0),'o') Grafico
  from LAVORATORE
 where Eta is NOT NULL
 order by Eta;
```

In questo esempio LPAD viene utilizzata come nel Capitolo 12, dove è stato mostrato l'albero genealogico delle mucche e dei tori di Talbot. In sostanza la colonna qui è costituita da una 'o' minuscola e viene riempita a sinistra con un certo numero aggiuntivo di 'o', fino alla larghezza massima determinata da ROUND(Eta/2,0).

Si noti che la scala della 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. I risultati di questa istruzione SQL sono mostrati nella Figura 15.2.

```
select Nome, Eta, LPAD('x',ROUND(Eta/2,0),' ') Grafico
  from LAVORATORE
```

| NOME | ETA | Eta | | | | | | |
|------------------|-----|--------------------------|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| HELEN BRANDT | 15 | 00000000 | | | | | | |
| WILLIAM SWING | 15 | 00000000 | | | | | | |
| DONALD ROLLO | 16 | 00000000 | | | | | | |
| DICK JONES | 18 | 0000000000 | | | | | | |
| PAT LAVAY | 21 | 000000000000 | | | | | | |
| BART SARJEANT | 22 | 000000000000 | | | | | | |
| ADAH TALBOT | 23 | 0000000000000 | | | | | | |
| PETER LAWSON | 25 | 00000000000000 | | | | | | |
| JOHN PEARSON | 27 | 000000000000000 | | | | | | |
| ANDREW DYE | 29 | 0000000000000000 | | | | | | |
| VICTORIA LYNN | 32 | 00000000000000000 | | | | | | |
| JED HOPKINS | 33 | 000000000000000000 | | | | | | |
| ROLAND BRANDT | 35 | 0000000000000000000 | | | | | | |
| GEORGE OSCAR | 41 | 00000000000000000000 | | | | | | |
| ELBERT TALBOT | 43 | 00000000000000000000 | | | | | | |
| GERHARDT KENTGEN | 55 | 0000000000000000000000 | | | | | | |
| WILFRED LOWELL | 67 | 000000000000000000000000 | | | | | | |

Figura 15.1 Diagramma a barre orizzontale dell'età di ogni persona.

| NOME | ETA | Eta | | | | | | |
|------------------|-----|-----|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| HELEN BRANDT | 15 | | x | | | | | |
| WILLIAM SWING | 15 | | x | | | | | |
| DONALD ROLLO | 16 | | x | | | | | |
| DICK JONES | 18 | | | x | | | | |
| PAT LAVAY | 21 | | | | x | | | |
| BART SARJEANT | 22 | | | | x | | | |
| ADAH TALBOT | 23 | | | | x | | | |
| PETER LAWSON | 25 | | | | x | | | |
| JOHN PEARSON | 27 | | | | x | | | |
| ANDREW DYE | 29 | | | | x | | | |
| VICTORIA LYNN | 32 | | | | x | | | |
| JED HOPKINS | 33 | | | | x | | | |
| ROLAND BRANDT | 35 | | | | x | | | |
| GEORGE OSCAR | 41 | | | | | x | | |
| ELBERT TALBOT | 43 | | | | | x | | |
| GERHARDT KENTGEN | 55 | | | | | | x | |
| WILFRED LOWELL | 67 | | | | | | x | |

Figura 15.2 Grafico dell'età di ogni persona.

```
where Eta is NOT NULL  
order by Eta;
```

Un altro modo per rappresentare l'età in un grafico è quello di basarsi sulla distribuzione invece che sulle persone. Innanzitutto viene creata una vista dividendo le età in decadi. Perciò 15, 16 e 18 diventano 10, le età da 20 a 29 diventano 20, quelle da 30 a 39 diventano 30 e così via:

```
create view INTERETA as  
select TRUNC(Eta,-1) Decade  
from LAVORATORE;
```

View created.

Poi viene creata un'intestazione della colonna, simile a quella precedente ma più corta e con incrementi di 1:

```
column Grafico Heading 'Conta| 1 |....5....0....5'-  
justify c  
column Grafico format a15  
column Persona Heading 'Int|Conta' format 9999
```

Questa istruzione SQL determina il numero di lavoratori la cui età è compresa in ciascuna delle decadi. Dal momento che i lavoratori con un'età sconosciuta non sono stati esclusi né qui né nella vista, in cima al diagramma appare una riga vuota per quelli la cui età è NULL (Figura 15.3).

```
select Decade, COUNT(Decade) Persona,  
      LPAD('o',COUNT(Decade),'o') Grafico  
   from INTERETA  
group by Decade;
```

Poiché COUNT ignora i valori NULL, non è stato possibile includere nella Figura 15.3 il numero di lavoratori la cui età è sconosciuta. Se invece di COUNT si fosse utilizzata COUNT(*), sarebbero state contate tutte le righe e il diagramma sarebbe stato simile a quello mostrato nella Figura 15.4. L'istruzione SQL che lo produce è la seguente:

```
select Decade, COUNT(*) Persona,  
      LPAD('o',COUNT(*),'o') Grafico  
   from INTERETA  
group by Decade;
```

15.3 Utilizzo di TRANSLATE

È opportuno ricordare che TRANSLATE converte i caratteri di una stringa in caratteri diversi, in base a un piano di sostituzione di *if* con *then*:

TRANSLATE(stringa,if,then)

| Conta | | | |
|--------|-------|-------------------|--------|
| Int | | 1 | 1 |
| DECADE | Conta |5.....0.....5 | |
| <hr/> | | | |
| 10 | | 4 | 0000 |
| 20 | | 6 | 000000 |
| 30 | | 3 | 000 |
| 40 | | 2 | 00 |
| 50 | | 1 | 0 |
| 60 | | 1 | 0 |

Figura 15.3 Distruzione dell'età in base alle decadi; i lavoratori la cui età è sconosciuta non vengono contati, per cui viene prodotta una riga vuota.

| Conta | | | |
|--------|-------|-------------------|--------|
| Int | | 1 | 1 |
| DECADE | Conta |5.....0.....5 | |
| <hr/> | | | |
| | | 2 | 00 |
| 10 | | 4 | 0000 |
| 20 | | 6 | 000000 |
| 30 | | 3 | 000 |
| 40 | | 2 | 00 |
| 50 | | 1 | 0 |
| 60 | | 1 | 0 |

Figura 15.4 Distribuzione dell'età, contando tutti i lavoratori.

Nella seguente istruzione SQL vengono sostituite le lettere in una frase. Ogni volta che viene rilevata una ‘T’ maiuscola, viene sostituita con un’altra ‘T’ maiuscola; in realtà non cambia nulla. Inoltre, ogni volta che viene rilevata una vocale maiuscola, viene sostituita con una 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. Perciò 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('ATTACCO ALLE VOCALI'
```

```
-----  
aTTaCCo aLLe VoCaLi
```

Eliminazione di caratteri

Che cosa succede se la stringa *if* è TAEIOU e la stringa *then* è soltanto T? Controllando la lettera ‘E’ (come nella parola “ALLE”), questa viene trovata nella posizione 3 di TAEIOU. Alla posizione 3 nella stringa *then* (che è costituita solo dalla lettera T) non vi è nulla, perciò al posto della ‘E’ non viene inserito nulla. Lo stesso procedimento viene applicato a tutte le vocali, dal momento che sono presenti nella stringa *if*, ma non in *then*. Quindi le vocali scompaiono, come mostrato di seguito:

```
select TRANSLATE('ATTACCO ALLE VOCALI','TAEIOU','T')  
from DUAL;
```

```
TRANSLATE('ATTACCO ALLE VOCAL'  
-----  
TTCC LL VCL
```

Questa caratteristica di TRANSLATE, la capacità di eliminare caratteri da una stringa, può essere molto utile per mettere a punto i dati. Si può ad esempio riprendere la tabella con i titoli di rivista descritti nel Capitolo 6:

```
select Titolo from RIVISTA;
```

```
TITOLO  
-----  
THE BARBERS WHO SHAVE THEMSELVES.  
"HUNTING THOREAU IN NEW HAMPSHIRE"  
THE ETHNIC NEIGHBORHOOD  
RELATIONAL DESIGN AND ENTHALPY  
"INTERCONTINENTAL RELATIONS."
```

Il metodo utilizzato nel Capitolo 6 per rimuovere i punti e i doppi apici consiste in una combinazione di LTRIM e RTRIM:

```
select LTRIM( RTRIM(Titolo,'."')) ,'''') from RIVISTA;
```

Lo stesso obiettivo può essere realizzato con una singola istruzione TRANSLATE:

```
select TRANSLATE(Titolo,'T".','T') TITOLO  
from RIVISTA;
```

```
TITOLO  
-----  
THE BARBERS WHO SHAVE THEMSELVES  
HUNTING THOREAU IN NEW HAMPSHIRE  
THE ETHNIC NEIGHBORHOOD  
RELATIONAL DESIGN AND ENTHALPY  
INTERCONTINENTAL RELATIONS
```

Rimozione di segni di dollaro e di virgole

Si supponga di avere un file di dati provenienti da un vecchio sistema di contabilità, salvato nel formato in cui è stato ricevuto, comprese virgole, punti decimali e persi-

no i segni di dollaro. Come è possibile risistemare i dati per poterli spostare in una colonna di numeri puri, dove non sono ammesse virgole e segni di dollaro?

Per regolare il formato numerico si potrebbe utilizzare la funzione TO_CHAR, oppure la funzione TRANSLATE.

La tabella VIRGOLA elenca semplicemente 11 righe di numeri formattati con virgole. Qui è riportata insieme con la traduzione che elimina le virgole:

```
select CarImporto, TRANSLATE(CarImporto,'1,$','1')
      from VIRGOLA;
```

| CARIMPORTO | TRANSLATE(CARIMPORTO) |
|------------------|-----------------------|
| \$0 | 0 |
| \$0.25 | 0.25 |
| \$1.25 | 1.25 |
| \$12.25 | 12.25 |
| \$123.25 | 123.25 |
| \$1,234.25 | 1234.25 |
| \$12,345.25 | 12345.25 |
| \$123,456.25 | 123456.25 |
| \$1,234,567.25 | 1234567.25 |
| \$12,345,678.25 | 12345678.25 |
| \$123,456,789.25 | 123456789.25 |

L'istruzione SQL ricerca un numero 1, una virgola o un segno di dollaro. Se trova un 1, lo sostituisce con 1. Se trova una virgola o un segno di dollaro, li elimina e al loro posto non inserisce nulla. È necessario che vi sia sempre almeno una lettera o un numero tradotto, qui un 1 e nell'esempio precedente una 'T', perché se in *then* non vi fosse alcun carattere reale, TRANSLATE non produrrebbe nulla:

```
select CarImporto, TRANSLATE(CarImporto,',','$','') from VIRGOLA;
```

| CARIMPORTO | TRANSLATE(CARIMPORTO) |
|------------------|-----------------------|
| \$0 | |
| \$0.25 | |
| \$1.25 | |
| \$12.25 | |
| \$123.25 | |
| \$1,234.25 | |
| \$12,345.25 | |
| \$123,456.25 | |
| \$1,234,567.25 | |
| \$12,345,678.25 | |
| \$123,456,789.25 | |

In questo caso la stringa *then* è vuota, senza alcun carattere, e la stringa *if* contiene solo due caratteri da eliminare, ma il comando non funziona. Infatti deve esservi almeno un carattere in entrambe le stringhe *if* e *then*. Se con questo metodo è possibile eliminare virgole e altri caratteri da una stringa, esiste anche un modo per inserire tali caratteri, ad esempio per introdurre delle virgole in un numero da visualizzare?

Per fare questo occorre utilizzare la funzione TO_CHAR. Questa funzione converte una stringa di numeri in una stringa di caratteri, consentendo di introdurre segni di dollaro e virgole o altri simboli. Per un elenco completo delle opzioni di formato per i numeri si può consultare la guida alfabetica di riferimento (Capitolo 37). Il seguente esempio mostra come formattare un numero con virgole.

Una maschera di formato comunica a ORACLE come formattare la stringa di caratteri che viene creata con questo comando:

```
select TO_CHAR(123456789,'999,999,999') TestVirgola
      from DUAL;
TESTVIRGOLA
-----
123,456,789
```

15.4 Copia e incollamento complessi

La tabella NOME contiene un elenco di nomi, ricevuto da un'azienda che commercializza elenchi di indirizzi o da un'altra applicazione. Il nome, il cognome e le iniziali sono tutti in una colonna:

```
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 per nome e cognome. Come si fa? La tecnica appresa nel Capitolo 6 utilizzava INSTR per individuare uno spazio e impiegava il numero restituito in una funzione SUBSTR per tagliare la parte compresa fino allo spazio. Qui è presentato un tentativo di fare lo stesso 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 6; uno di essi 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 un'espressione come questa:

```
DECODE(INSTR(Nome, ' '), 0, 99, INSTR(Nome, ' '))
```

Si ricordi che la sintassi di DECODE è:

```
DECODE(valore, if1, then1[, if2, then2, if3, then3]..., else)
```

In questo esempio DECODE controlla il valore di INSTR(Nome,''). Se è uguale a 0 sostituisce 99, altrimenti restituisce il valore di default, che coincide con INSTR(Nome,''). La scelta di 99 come sostituto è relativamente 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 punto iniziale 1 fino al punto finale o alla fine della stringa. 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 può di nuovo utilizzare INSTR per ricercare lo spazio e impiegare il numero restituito (+1) come punto iniziale per SUBSTR. Non è necessario alcun punto finale, perché si vuole arrivare fino alla fine del nome. Ecco che cosa succede:

```
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 è quella di utilizzare tre funzioni INSTR, cercando di seguito la prima, la seconda o la terza ricorrenza di uno spazio nel nome. Ognuna di queste funzioni restituisce la locazione dove ha trovato uno spazio, oppure uno zero se non ne ha trovato alcuno. La funzione GREATEST preleva quindi 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 poteva essere utilizzata in modo simile anche nell'esempio precedente al posto di DECODE). Vi è 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 in Nome deve essere iniziata dalla posizione finale, spostandosi all'indietro, o da destra verso sinistra. Quando trova lo spazio, INSTR restituisce la sua posizione, contando da sinistra come al solito. Tutti i **-1** inducono 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 **+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 e perciò 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 di default (che va automaticamente fino alla fine). La posizione finale viene trovata utilizzando il seguente comando:

```
DECODE(INSTR(Nome,' '),0,0,LENGTH(Nome))
```

Il comando cerca la posizione di uno spazio in Nome. Se la posizione è zero, restituisce zero; altrimenti restituisce LENGTH di Nome. Per VALDO, DECODE produce zero come posizione finale per SUBSTR, così non viene visualizzato nulla.

Per qualsiasi altro nome, poiché vi è sicuramente uno spazio da qualche parte, la posizione finale di SUBSTR diventa LENGTH di Nome, così viene visualizzato l'ultimo nome per intero.

Questo comando è simile al comando DECODE utilizzato per estrarre il primo nome, a parte il fatto che il valore 99 impiegato in quel caso è stato sostituito da LENGTH(Nome), che funziona sempre, mentre 99 potrebbe fallire per un nome più lungo 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 dalla parte destra di una stringa tutto quello che è specificato in *set*, 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. Ora occorre creare la tabella con le colonne del nome e del cognome (tutti i dettagli per la creazione delle tabelle sono riportati nel Capitolo 17).

```
create table DUENOMI(
PrimoNome VARCHAR2(25),
Cognome VARCHAR2(25)
);
```

Table created.

Per riempire la tabella con i nomi e cognomi della tabella NOME si deve utilizzare insert con una sottoquery:

```
insert into DUENOMI (PrimoNome, 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;
```

| PRIMONOME | COGNOME |
|-----------|----------|
| HORATIO | NELSON |
| VALDO | |
| MARIE | MEDICIS |
| FLAVIUS | JOSEPHUS |
| EDYTHE | GAMMIERE |

Si può utilizzare una tecnica simile per estrarre le iniziali e applicare questi metodi anche altrove, ad esempio per indirizzi, descrizioni di prodotti, nomi di aziende e così via.

Quando si spostano i dati da un vecchio sistema a uno nuovo, spesso occorre eseguire una difficile riformattazione. In SQL esistono delle funzionalità che facilitano il compito, ma richiedono un certa conoscenza di come operano le funzioni e un po' di attenzione per evitare problemi come quelli mostrati qui.

15.5 Conteggio delle ricorrenze di stringhe in stringhe più lunghe

Si può utilizzare una combinazione delle funzioni LENGTH e REPLACE per determinare 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. Perciò:

```
REPLACE('ADAH', 'A', 'BLAH')
```

esamina la stringa ‘ADAH’. Ovunque trova una ‘A’, la sostituisce con ‘BLAH’. Quindi la funzione mostrata in questo esempio restituisce la stringa ‘BLAHD-BLAHH’.

Si può utilizzare la funzione REPLACE per eliminare dei caratteri da una stringa. Ad esempio, si può sostituire la stringa di caratteri che si sta cercando con un valore NULL. Perciò:

```
REPLACE('GEORGE', 'GE', NULL)
```

restituisce 'OR'. Le due distinte ricorrenze di 'GE' in 'GEORGE' sono state tutte poste a NULL.

Si può utilizzare la funzione REPLACE per determinare quante volte una stringa (come 'GE') viene trovata in una stringa più lunga (come 'GEORGE'). Innanzitutto occorre sostituire la stringa con valori NULL:

```
select REPLACE('GEORGE', 'GE', NULL)
  from DUAL;
```

Il risultato di questa query è 'OR'. Più importante è la lunghezza del risultato, che fornisce informazioni su quante volte la stringa è stata trovata. La seguente query calcola quanto è lunga la stringa risultante:

```
select LENGTH(REPLACE('GEORGE', 'GE', NULL))
  from DUAL;
```

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 la differenza per la lunghezza della stringa ricercata, si ottiene come risultato il numero di ricorrenze della stringa ricercata:

```
select (LENGTH('GEORGE') -
        LENGTH(REPLACE('GEORGE', 'GE', NULL))) /
        LENGTH('GE')
  from DUAL;
```

Questo esempio utilizza stringhe invece di colonne di caratteri per rendere più semplice la comprensione; nelle applicazioni reali si dovrebbe sostituire la stringa originaria ('GEORGE') con un 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. Perciò la differenza fra la stringa originale e quella accorciata è di quattro caratteri. Dividendo i quattro caratteri per la lunghezza della stringa di ricerca (due caratteri) si deduce che la stringa è stata trovata due volte.

Il solo problema con questo metodo si ha quando la stringa ricercata è di zero caratteri (dal momento che non è possibile dividere per zero). La ricerca di una stringa di lunghezza zero può indicare un problema nella logica dell'applicazione che l'iniziata.

15.6 Variabili e sostituzioni registrate

Si supponga di dover realizzare un report che deve essere utilizzato da funzionari bancari per determinare quando matureranno i buoni che hanno emesso. La difficoltà è costituita dall'avvicinarsi dell'anno 2000, per cui alcuni buoni matureranno

dopo il 1999. Innanzitutto occorre creare un report con tutti i titoli e le date di maturazione:

```
ttitle 'Maturazione buoni'

select Conto, Importo, DataMaturazione
  from BUONO;

Sat Apr 1          page    1
                  Maturazione buoni

      CONTO      IMPORTO DATAMATUR
-----  -----
      573334      10000 15-JAN-09
      677654      25000 15-JAN-01
      976032      10000 15-JAN-95
     275031      10000 15-JAN-97
     274598      20000 15-JAN-99
     538365      45000 15-JAN-01
     267432      16500 15-JAN-04
```

Qui sono rappresentati vari conti e anni. Si è interessati in particolare a quelli che matureranno il 15 gennaio del 2001, perciò la query viene modificata in questo modo:

```
select Conto, Importo
  from BUONO
 where DataMaturazione = '15-JAN-01';

no rows selected
```

Sfortunatamente ORACLE presuppone che qualsiasi anno di due cifre inserito sia nel secolo corrente, per cui ritiene che si stia facendo riferimento all'anno 1901. Un modo per risolvere questo problema consiste nel cambiare la query in maniera che TO_DATE possa accettare un anno di quattro cifre:

```
select Conto, Importo
  from BUONO
 where DataMaturazione = TO_DATE('15-JAN-2001','DD-MON-YYYY');
```

```
Sat Apr 1          page    1
                  Maturazione buoni

      CONTO      IMPORTO
-----  -----
      677654      25000
      538365      45000
```

Però questo report deve essere eseguito da un funzionario di banca che potrebbe non conoscere SQL o non sapere come scrivere le query. Per facilitare il compito del funzionario si può utilizzare una variabile e il comando accept:

```
set verify off
set echo off
```

```

accept Test prompt 'Inserire data nel formato GG/MM/AAAA: '

spool maturity.lst
select Conto, Importo
  from BUONO
 where DataMaturazione = TO_DATE('&Test','DD/MM/YYYY');

spool off

```

Questo codice, quando viene inserito in un file di avvio ed eseguito, chiede al funzionario quanto segue:

Inserire data nel formato GG/MM/AAAA:

Esiste un approccio alternativo che consente anche di inserire anni di due cifre e ampliarli in quattro cifre con il secolo corretto. Si potrebbe naturalmente utilizzare il formato “RR” per l’anno; questo formato interpreta tutti gli anni di due cifre prima di 50 come appartenenti al ventunesimo secolo. Ma si supponga di voler trattare in questo modo solo gli anni fino a 10. Per ottenere questo effetto si potrebbe impiegare DECODE o utilizzare una caratteristica aggiuntiva di SQLPLUS e dei file di avvio. Si osservi il file di avvio riportato nella Figura 15.5.

Un approccio anche migliore consiste nell’impostare il formato a quattro cifre per l’anno nel file init.ora o per una sessione con l’istruzione alter session.

Al punto 1 tutti i controlli che potrebbero produrre messaggi non pertinenti nella visualizzazione vengono disattivati.

Al punto 2 si richiede all’utente di inserire una data con un anno di due cifre, nel formato DD/MM/ YY. Si supponga che l’utente abbia inserito 15/01/01.

Al punto 3 il file di avvio inizia a registrare i risultati in un altro file chiamato t.sql. Questi risultati sono mostrati nella Figura 15.6.

Al punto 4 viene eseguita un’istruzione select che confronta le ultime due cifre della data, 01, con l’anno di riferimento, 90, utilizzando la tabella fittizia DUAL, che ha una sola riga. Poiché 01 non è maggiore o uguale a 90, questa istruzione select non produce alcun record. Poiché feedback è off, non appare alcun messaggio “no row selected”.

Al punto 5 viene eseguita un’istruzione select quasi simile, ma poiché 01 è minore di 90 questa ha successo e aggiunge 2000 a 01. I risultati di questa select sono visibili al punto A nella Figura 15.6. Si noti come il letterale ‘define YYYY=’ viene ora combinato con il risultato del calcolo, 2001.

Al punto 6 la prima parte della data è tagliata dalla variabile Test e definita uguale alla variabile GiornoMese. L’effetto è mostrato al punto B nella Figura 15.6.

Una serie di prompt al punto 7 ora mostra questo messaggio:

```

set feedback on
set heading on
ttitle on

```

Poiché il file di avvio principale viene ancora registrato in t.sql, la parola prompt ha come effetto la scrittura di tutto il testo che segue nel file di registrazione, e ciò che compare in t.sql al punto C. prompt non seguito da alcun testo ha scritto una riga vuota.

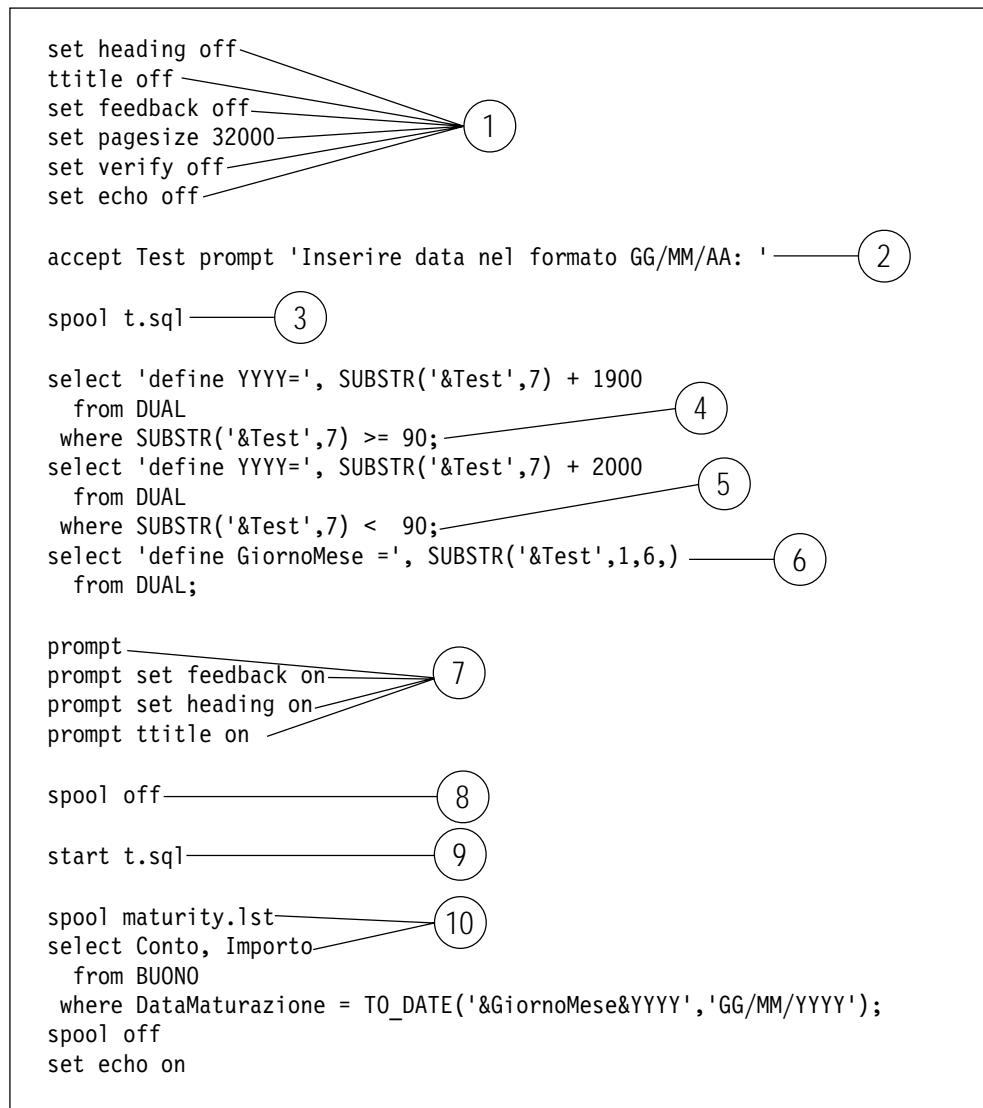


Figura 15.5 File di avvio con un utilizzo complesso delle variabili.

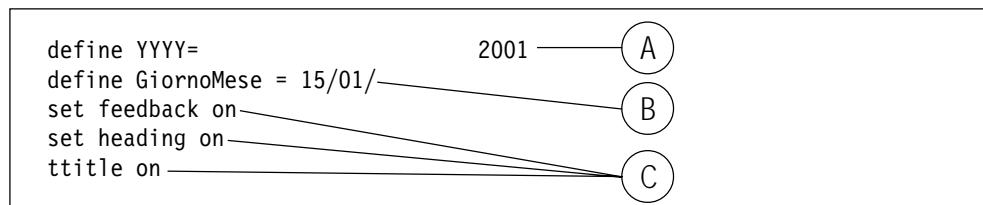


Figura 15.6 Il file di avvio registrato T.SQL.

Al punto 8 la registrazione in t.sql è disattivata. Quindi il file viene chiuso con le informazioni mostrate nella Figura 15.6.

Al punto 9 il file t.sql viene avviato. Tutti i comandi della Figura 15.6 vengono ora eseguiti. YYYY viene definito come 2001. GiornoMese è definito come 15/01/. feedback, heading e title vengono riportati al loro stato iniziale.

Al punto 10 inizia la registrazione sul file di output maturity.lst e viene eseguita la seguente istruzione select:

```
select Conto, Importo
  from BUONO
 where DataMaturazione = TO_DATE('&GiornoMese&YYYY', 'DD/MM/YYYY');
```

Poiché &GiornoMese e &YYYY vengono concatenate, si ottiene 15/01/2001 e la query viene eseguita, producendo il seguente report nel file maturity.lst.

```
Sat Apr 1                               page    1
                                         Maturazione buoni

      CONTO        IMPORTO
-----
 677654          25000
 538365          45000

2 rows selected.
```

Alcune informazioni in più sulle variabili

Il comando accept induce SQLPLUS ad assegnare alla variabile il valore fornito; questo può essere fatto 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 accept nella guida alfabetica di riferimento del Capitolo 37).

È anche possibile passare argomenti al file di avvio incorporando variabili numerate nelle istruzioni select (invece che variabili con nomi).

Per selezionare tutte le transazioni di Talbot tra due date, si potrebbe scrivere un'istruzione select simile a questa:

```
select * from REGISTRO
  where DataAzione BETWEEN '&1' AND '&2';
```

Poi si dovrebbe avviare la query in questo modo:

```
start ledger.sql 01-JAN-01 31-DEC-01
```

Il file di avvio ledger.sql dovrebbe iniziare con 01-JAN-01 sostituito a &1 e 31-DEC-01 sostituito a &2. Come con altre variabili, i tipi di dati carattere e DATE nell'istruzione SQL devono essere racchiusi tra apici singoli, tranne quando ogni argomento che segue la parola start è una parola singola, senza spazi.

Le sostituzioni di variabili non sono limitate solo all'istruzione SQL. Il file di avvio può utilizzare variabili anche per comandi SQLPLUS.

Si è visto in precedenza che le variabili possono essere concatenate semplicemente ponendole una a fianco dell'altra. Si può concatenare una variabile con una costante utilizzando il punto (.).

```
select sum(Importo)
      from REGISTRO
     where DataAzione BETWEEN '01-&Mese.-01'
                           AND LAST_DAY(TO_DATE('01-&Mese.-01'));
```

Questa istruzione select effettua la query solo per un mese e poi 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 segno &.*

Comandi set

Normalmente una variabile è denotata dalla ‘e’ commerciale (&), ma questo simbolo può essere cambiato utilizzando l’istruzione set define seguita dal simbolo di un solo carattere che si preferisce per indicare le variabili.

set escape definisce un carattere che può essere posto immediatamente davanti alla ‘e’ commerciale (o a un altro simbolo definito) in modo che SQLPLUS possa trattare questo simbolo come un letterale, invece che per indicare una variabile.

set concat può sostituire l’operatore di concatenazione di default per le variabili, che è il punto (.), con un altro simbolo di una sola lettera.

Questa concatenazione di variabili si utilizza in aggiunta all’operatore di concatenazione SQL, e separatamente da esso, che di solito 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 trattata come un letterale; ad esempio, &Test è trattata come la parola letterale &Test e non come il valore in essa definito. Le variabili in SQLPLUS, comunque, vengono ancora sostituite come prima.

•
•
•
•
• Capitolo 16

• La funzione DECODE

- •
• 16.1 **if, then, else**
• 16.2 **Esempio: datazione di fatture**
• 16.3 **Trasposizione di una tabella**
• 16.4 **Utilizzo di MOD in DECODE**
• 16.5 **order by e RowNum**
• 16.6 **Colonne e calcoli in then ed else**
• 16.7 **Maggiore, minore e uguale in DECODE**

DECODE è senza dubbio la funzione più potente nel linguaggio SQL di ORACLE. È una delle varie estensioni aggiunte da ORACLE al linguaggio SQL standard e non è ancora disponibile in quello fornito da altri, per cui si potrebbe sostenere che l'SQL di ORACLE non è standard. Anche se questo è vero, si tratta una critica che non ha senso, infatti le implementazioni di SQL prive di DECODE risultano impoverite dalla sua mancanza. In questo capitolo vengono esaminati alcuni modi in cui può essere utilizzata DECODE.

Questo è un capitolo avanzato e l'argomento è difficile. Gran parte degli argomenti illustrati qui non sono necessari per la grande maggioranza dei report e possono essere tralasciati se vanno al di là delle proprie esigenze. In effetti si tratta di materiale importante soprattutto per complesse operazioni di realizzazione di report e di programmazione.

16.1 **if, then, else**

Nella programmazione e nella logica, una comune costruzione di un problema è basata sul modello *if, then, else*. Ad 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, segue un certo risultato; se non corrisponde a nessuno di questi giorni, segue un altro risultato. DECODE rispecchia questo tipo di logica.

Nel Capitolo 9 è stata fornita un'introduzione, nella quale è stata mostrata la struttura di base e l'utilizzo di DECODE.

La sintassi di questo comando è la seguente:

```
DECODE(valore,if1,then1,if2,then2,if3,then3,...,else)
```

valore rappresenta una 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 meno un altro e così via. *valore* viene controllato per 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* e *else* può essere una colonna o il risultato di una funzione o di un calcolo.

16.2 Esempio: datazione di fatture

Si supponga che Talbot abbia una tabella o una vista chiamata FATTURA che contiene le fatture non pagate, le loro date e i nomi dei relativi clienti. Il 15 dicembre 1901 la tabella potrebbe contenere i risultati mostrati nella Figura 16.1.

```
column NomeCliente format a14
column DataFattura heading 'Data|Fattura'

select NomeCliente, DataFattura, Importo
      from FATTURA;
```

| NOMECLIENTE | Data | |
|---------------|-----------|---------|
| | Fattura | IMPORTO |
| ELBERT TALBOT | 23-OCT-01 | 5.03 |
| JOHN PEARSON | 09-NOV-01 | 2.02 |
| DICK JONES | 12-SEP-01 | 11.12 |
| GENERAL STORE | 09-NOV-01 | 22.10 |
| ADAH TALBOT | 17-NOV-01 | 8.29 |
| GENERAL STORE | 01-SEP-01 | 21.32 |
| ADAH TALBOT | 15-NOV-01 | 7.33 |
| GENERAL STORE | 04-OCT-01 | 8.42 |
| KAY WALLBOM | 04-OCT-01 | 1.43 |
| JOHN PEARSON | 13-OCT-01 | 12.41 |
| DICK JONES | 23-OCT-01 | 4.49 |
| GENERAL STORE | 23-NOV-01 | 40.36 |
| GENERAL STORE | 30-OCT-01 | 7.47 |
| MORRIS ARNOLD | 03-OCT-01 | 3.55 |
| ROLAND BRANDT | 22-OCT-01 | 13.65 |
| MORRIS ARNOLD | 21-SEP-01 | 9.87 |
| VICTORIA LYNN | 09-OCT-01 | 8.98 |
| GENERAL STORE | 22-OCT-01 | 17.58 |

Figura 16.1 Clienti e importi di fatture da pagare.

Esaminando questo elenco, Talbot comprende che molte fatture sono insolute da molto tempo, per cui intende analizzare quanto è grave il problema. Il primo tentativo è mostrato nella Figura 16.2.

```
select NomeCliente, DataFattura, Importo
  from FATTURA
 order by DataFattura;
```

| NOMECLIENTE | Data Fattura | IMPORTO |
|---------------|-----------------|---------|
| GENERAL STORE | 01-SEP-01 | 21.32 |
| DICK JONES | 12-SEP-01 | 11.12 |
| MORRIS ARNOLD | 21-SEP-01 | 9.87 |
| MORRIS ARNOLD | 03-OCT-01 | 3.55 |
| GENERAL STORE | 04-OCT-01 | 8.42 |
| KAY WALLBOM | 04-OCT-01 | 1.43 |
| VICTORIA LYNN | 09-OCT-01 | 8.98 |
| JOHN PEARSON | 13-OCT-01 | 12.41 |
| ROLAND BRANDT | 22-OCT-01 | 13.65 |
| GENERAL STORE | 22-OCT-01 | 17.58 |
| ELBERT TALBOT | 23-OCT-01 | 5.03 |
| DICK JONES | 23-OCT-01 | 4.49 |
| GENERAL STORE | 30-OCT-01 | 7.47 |
| JOHN PEARSON | 09-NOV-01 | 2.02 |
| GENERAL STORE | 09-NOV-01 | 22.10 |
| ADAH TALBOT | 15-NOV-01 | 7.33 |
| ADAH TALBOT | 17-NOV-01 | 8.29 |
| GENERAL STORE | 23-NOV-01 | 40.36 |

Figura 16.2 Clienti ordinati in base alla data delle fatture.

Questa vista è più utile, ma non produce alcuna reale analisi per determinare la gravità del problema di Talbot. Perciò viene creata un'espressione DECODE per stabilire come queste fatture e importi sono distribuiti nel tempo, come mostra la Figura 16.3.

Per analizzare la logica utilizzata in ciascuna espressione DECODE, è sufficiente considerare solo l'espressione per le fatture emesse da più di novanta giorni.

```
DECODE(TRUNC((Giorno-DataFattura)/30),3,Importo,NULL) NOVANTA
```

Giorno è una data: 15 dicembre 1901. La data è ottenuta da una piccola tabella, GIORNO, creata appositamente per il report del giorno e contenente solo una data. Se una query di questo tipo dovesse essere sempre aggiornata al momento attuale, si potrebbe utilizzare anche SysDate, ma Giorno è più utile quando un report deve essere riferito a un determinato giorno, invece che a oggi (si sarebbe anche potuto utilizzare una variabile nel file di avvio, con la funzione TO_DATE).

```

select NomeCliente,
       TRUNC((Giorno-DataFattura)/30) GIORNI,
       DECODE(TRUNC((Giorno-DataFattura)/30),0,Importo, NULL) OGGI,
       DECODE(TRUNC((Giorno-DataFattura)/30),1,Importo, NULL) TRENTA,
       DECODE(TRUNC((Giorno-DataFattura)/30),2,Importo, NULL) SESSANTA,
       DECODE(TRUNC((Giorno-DataFattura)/30),3,Importo, NULL) NOVANTA
  from FATTURA, GIORNO
 order by DataFattura;

```

| NOMECLIENTE | GIORNI | OGGI | TRENTA | SESSANTA | NOVANTA |
|---------------|--------|-------|--------|----------|---------|
| GENERAL STORE | 3 | | | | 21.32 |
| DICK JONES | 3 | | | | 11.12 |
| MORRIS ARNOLD | 2 | | | 9.87 | |
| MORRIS ARNOLD | 2 | | | 3.55 | |
| GENERAL STORE | 2 | | | 8.42 | |
| KAY WALLBOM | 2 | | | 1.43 | |
| VICTORIA LYNN | 2 | | | 8.98 | |
| JOHN PEARSON | 2 | | | 2.41 | |
| ROLAND BRANDT | 1 | 13.65 | | | |
| GENERAL STORE | 1 | 17.58 | | | |
| ELBERT TALBOT | 1 | 5.03 | | | |
| DICK JONES | 1 | 4.49 | | | |
| GENERAL STORE | 1 | 7.47 | | | |
| JOHN PEARSON | 1 | 2.02 | | | |
| GENERAL STORE | 1 | 22.10 | | | |
| ADAH TALBOT | 1 | 7.33 | | | |
| ADAH TALBOT | 0 | 8.29 | | | |
| GENERAL STORE | 0 | 40.36 | | | |

Figura 16.3 DECODE utilizzata per disporre gli importi dovuti rispetto al tempo trascorso.

Il valore che deve essere decodificato è un calcolo. Parte dalla “T” dopo l’apertura della prima parentesi e termina con la prima virgola. In questa istruzione DECODE, per ogni riga viene sottratta DataFattura alla data di Giorno. Il risultato è il numero di giorni trascorsi dalla data della fattura. Questo intervallo viene poi diviso per 30, per ottenere il numero di periodi di 30 giorni dalla data della fattura.

Per intervalli che non sono multipli esatti di 30, questo non è un numero intero, perciò viene troncato, per fare in modo che il risultato sia sempre un numero intero. Qualsiasi data che precede il 15 dicembre di meno di 30 giorni produce 0. Una data compresa tra 30 e 59 giorni produce 1. Una data compresa tra 60 e 89 produce 2. Una data compresa tra 90 e 119 produce 3. Il numero 0, 1, 2 o 3 corrisponde a *valore* nell’istruzione DECODE.

È opportuno esaminare di nuovo l’ultima istruzione DECODE. Dopo la prima virgola vi è un 3. Questo è il test *if*. Se *valore* è 3, allora l’intera istruzione DECODE in questa riga sarà equivalente a Importo. Se *valore* è diverso da 3 (ovvero meno di 90 giorni o più di 119), DECODE sarà uguale a NULL.

Se si confrontano le date delle fatture nella Figura 16.2 con gli importi nella colonna NOVANTA della Figura 16.3, si vede come funziona questa logica.

Raggruppamento di clienti

Come passaggio successivo nell'analisi, si desidera raggruppare tutti i clienti, con gli importi da pagare, divisi per periodi. Per fare questo occorre aggiungere un semplice comando `order by` all'ultima istruzione SQL, come mostrato nella Figura 16.4.

Sfortunatamente, nella visualizzazione si ottiene una riga per ogni fattura. Sarebbe più utile sommare gli importi dovuti in modo che ogni cliente occupi una sola riga, con gli importi disposti in base al periodo.

```
select NomeCliente,
       TRUNC((Giorno-DataFattura)/30) GIORNI,
       DECODE(TRUNC((Giorno-DataFattura)/30),0,Importo, NULL) OGGI,
       DECODE(TRUNC((Giorno-DataFattura)/30),1,Importo, NULL) TRENTA,
       DECODE(TRUNC((Giorno-DataFattura)/30),2,Importo, NULL) SESSANTA,
       DECODE(TRUNC((Giorno-DataFattura)/30),3,Importo, NULL) NOVANTA
  from FATTURA, GIORNO
 order by NomeCliente, DataFattura;
```

| NOMECLIENTE | GIORNI | OGGI | TRENTA | SESSANTA | NOVANTA |
|---------------|--------|-------|--------|----------|---------|
| ADAH TALBOT | 1 | | 7.33 | | |
| ADAH TALBOT | 0 | 8.29 | | | |
| DICK JONES | 3 | | | | 11.12 |
| DICK JONES | 1 | | 4.49 | | |
| ELBERT TALBOT | 1 | | 5.03 | | |
| GENERAL STORE | 3 | | | | 21.32 |
| GENERAL STORE | 2 | | | 8.42 | |
| GENERAL STORE | 1 | | 17.58 | | |
| GENERAL STORE | 1 | | 7.47 | | |
| GENERAL STORE | 1 | | 22.10 | | |
| GENERAL STORE | 0 | 40.36 | | | |
| JOHN PEARSON | 2 | | | 12.41 | |
| JOHN PEARSON | 1 | | 2.02 | | |
| KAY WALLBOM | 2 | | | 1.43 | |
| MORRIS ARNOLD | 2 | | | 9.87 | |
| MORRIS ARNOLD | 2 | | | 3.55 | |
| ROLAND BRANDT | 1 | | 13.65 | | |
| VICTORIA LYNN | 2 | | | 8.98 | |

Figura 16.4 Fatture di clienti raggruppate insieme.

Questo viene realizzato con una vista:

```
create or replace view SCADUTA as
select NomeCliente,
       SUM(DECODE(TRUNC((Giorno-DataFattura)/30),0,Importo,NULL)) OGGI,
       SUM(DECODE(TRUNC((Giorno-DataFattura)/30),1,Importo,NULL)) TRENTA,
       SUM(DECODE(TRUNC((Giorno-DataFattura)/30),2,Importo,NULL)) SESSANTA,
       SUM(DECODE(TRUNC((Giorno-DataFattura)/30),3,Importo,NULL)) NOVANTA
       SUM(Importo) TOTAL
  from FATTURA, GIORNO
 group by NomeCliente;
```

La vista è seguita da una semplice query con le intestazioni delle colonne e le istruzioni compute e break on utilizzate per mostrare i totali per colonna. Per ottenere i totali generali viene impiegata la pseudocolonna User (una colonna che non è realmente nella tabella, come SysDate), invece dell'opzione Report di break on spiegata nel Capitolo 13 (semplicemente per mostrare una tecnica alternativa). La query è mostrata nella Figura 16.5.

Ogni cliente è consolidato in una sola riga, con gli importi dovuti per ogni periodo e quello totale. Probabilmente si è notata un'interessante abbreviazione nell'istruzione select mostrata nella Figura 16.5: un asterisco (*) dopo select consente di ottenere tutte le colonne regolari in una tabella o vista. Qui, oltre alle colonne reali, si desidera anche la pseudocolonna User. Per utilizzare il metodo abbreviato e ottenere le colonne aggiuntive, l'asterisco deve essere preceduto dal nome della tabella e da un punto. L'asterisco, se deve essere seguito dal nome di un'altra pseudocolonna o colonna reale, non può stare da solo. In altre parole, questa istruzione:

```
select SCADUTA.* , User
```

è identica a questa:

```
select NomeCliente, Oggi, Trenta, Sessanta, Novanta, Totale, User
```

User può essere richiesta in una query per mostrare chi ha eseguito il report in title o btitle. Nell'esempio della Figura 16.5 è utilizzata in un maniera più inusuale: produce lo stesso effetto di break on Report. In altre parole, il codice seguente:

```
column User noprint
column Oggi heading 'CORRENTE'
compute sum of Oggi Trenta Sessanta Novanta Totale on User
break on User skip 1
```

```
select SCADUTA.* , User
      from SCADUTA;
```

produce esattamente lo stesso risultato di questo:

```
column Oggi heading 'CORRENTE'
compute sum of Oggi Trenta Sessanta Novanta Totale on Report
break on Report skip 1
```

```
select *
      from SCADUTA;
```

```

column User noprint
column OGGI heading 'CORRENTE'
compute sum of OGGI TRENTA SESSANTA NOVANTA Total on User
break on User skip 1

```

```

select SCADUTA.* , User
      from SCADUTA;

```

| NOMECLIENTE | CORRENTE | TRENTA | SESSANTA | NOVANTA | TOTAL |
|---------------|----------|--------|----------|---------|--------|
| ADAH TALBOT | 8.29 | 7.33 | | | 15.62 |
| DICK JONES | | 4.49 | | 11.12 | 15.61 |
| ELBERT TALBOT | | 5.03 | | | 5.03 |
| GENERAL STORE | 40.36 | 47.15 | 8.42 | 21.32 | 117.25 |
| JOHN PEARSON | | 2.02 | 12.41 | | 14.43 |
| KAY WALLBOM | | | 1.43 | | 1.43 |
| MORRIS ARNOLD | | | 13.42 | | 13.42 |
| ROLAND BRANDT | | 13.65 | | | 13.65 |
| VICTORIA LYNN | | | 8.98 | | 8.98 |
| | 48.65 | 79.67 | 44.66 | 32.44 | 205.42 |

Figura 16.5 Clienti consolidati ciascuno in una sola riga.

È opportuno confrontare la presenza di Report nella seconda di queste due istruzioni SQL con User nella prima. La seconda è la normale tecnica di creazione di un report con totali. Il primo metodo produce gli stessi risultati. Funziona perché la pseudocolonna User, che contiene sempre il nome utente, non cambia da riga a riga. break on non rileva un cambiamento in User finché non si è superata l'ultima riga e non si è terminata la query. Inoltre, se si applica alla colonna User l'opzione noprint, questa non viene mostrata nel report. Questo era il metodo utilizzato prima che in ORACLE fosse disponibile l'opzione Report per break on e compute, e potrebbe essere utile per altre applicazioni. Qui viene mostrato non come alternativa consigliata a Report (infatti non lo è), ma soltanto per presentare un'altra tecnica utilizzabile.

16.3 Trasposizione di una tabella

Una tabella è costituita da colonne e righe. Le colonne sono predefinite: hanno un particolare nome e tipo di dati e sono in numero limitato. Le righe sono diverse: il loro numero varia da zero a diversi milioni e il valore per una colonna può cambiare da riga a riga e non è predefinito, in pratica può essere qualsiasi cosa, purché corrisponda al tipo di dati.

Questo è il caso generico, ovvero vale per le tabelle in generale. Però le tabelle sono spesso molto più ristrette, stabili e definite. Una tabella con i nomi delle festività, ad esempio, non richiede nuove righe molto spesso.

Altre tabelle sono un po' più variabili, ma rimangono stabili per lunghi periodi di tempo; in alcuni casi i valori di determinate colonne, come i nomi dei clienti principali, possono rimanere invariati. Casi come questo forniscono l'opportunità di utilizzare DECODE in un modo più inusuale: per trasporre una tabella, ovvero trasformare alcuni valori di riga in colonne. In questo esempio, la tabella FATTURA viene trasposta in una vista chiamata ClientePerData. Per ogni DataFattura, ogni NomeCliente diventa una colonna di numeri che contiene l'importo della fattura per quella data:

```
create or replace view ClientePerData as
select DataFattura,
       SUM(DECODE(NomeCliente,'ADAH TALBOT', Importo, 0)) AdahTalbot,
       SUM(DECODE(NomeCliente,'ELBERT TALBOT', Importo, 0)) ElbertTalbot,
       SUM(DECODE(NomeCliente,'VICTORIA LYNN', Importo, 0)) VictoriaLynn,
       SUM(DECODE(NomeCliente,'JOHN PEARSON', Importo, 0)) JohnPearson,
       SUM(DECODE(NomeCliente,'DICK JONES', Importo, 0)) DickJones,
       SUM(DECODE(NomeCliente,'GENERAL STORE', Importo, 0)) GeneralStore,
       SUM(DECODE(NomeCliente,'KAY WALLBOM', Importo, 0)) KayWallbom,
       SUM(DECODE(NomeCliente,'MORRIS ARNOLD', Importo, 0)) MorrisArnold,
       SUM(DECODE(NomeCliente,'ROLAND BRANDT', Importo, 0)) RolandBrandt
  from FATTURA
 group by DataFattura;
```

Quando la vista viene rappresentata, appare in questo modo:

```
describe CLIENTEPERDATA
```

| Name | Null? | Type |
|--------------|-------|--------|
| DATAFATTURA | | DATE |
| ADAHTALBOT | | NUMBER |
| ELBERTTALBOT | | NUMBER |
| VICTORIALYNN | | NUMBER |
| JOHNPEARSON | | NUMBER |
| DICKJONES | | NUMBER |
| GENERALSTORE | | NUMBER |
| KAYWALLBOM | | NUMBER |
| MORRISARNOLD | | NUMBER |
| ROLANDBRANDT | | NUMBER |

Effettuando una query su questa vista si può vedere una riga per ogni data, con uno zero o un importo dovuto sotto ogni colonna. Si noti che in questa query sono stati inclusi solo cinque clienti, semplicemente per rendere più semplice la lettura (Figura 16.6).

Su questa tabella si potrebbe costruire un'altra vista, magari sommando i totali per cliente per tutte le date e consolidando orizzontalmente tutti gli importi per tutti i clienti. Questo metodo trasforma calcoli orizzontali in verticali e viceversa ed è utile soprattutto in complessi calcoli di array e matrici.

```
column DataFattura heading 'Data|Fattura'

select DataFattura, ElbertTalbot, GeneralStore, DickJones, KayWallbom,
       RolandBrandt
  from CLIENTEPERDATA;
```

| Data Fattura | ELBERTTALBOT | GENERALSTORE | DICKJONES | KAYWALLBOM | ROLANDBRANDT |
|-----------------|--------------|--------------|-----------|------------|--------------|
| 01-SEP-01 | 0 | 21.32 | 0 | 0 | 0 |
| 12-SEP-01 | 0 | 0 | 11.12 | 0 | 0 |
| 21-SEP-01 | 0 | 0 | 0 | 0 | 0 |
| 03-OCT-01 | 0 | 0 | 0 | 0 | 0 |
| 04-OCT-01 | 0 | 8.42 | 0 | 1.43 | 0 |
| 09-OCT-01 | 0 | 0 | 0 | 0 | 0 |
| 13-OCT-01 | 0 | 0 | 0 | 0 | 0 |
| 22-OCT-01 | 0 | 17.58 | 0 | 0 | 13.65 |
| 23-OCT-01 | 5.03 | 0 | 4.49 | 0 | 0 |
| 30-OCT-01 | 0 | 7.47 | 0 | 0 | 0 |
| 09-NOV-01 | 0 | 22.10 | 0 | 0 | 0 |
| 15-NOV-01 | 0 | 0 | 0 | 0 | 0 |
| 17-NOV-01 | 0 | 0 | 0 | 0 | 0 |
| 23-NOV-01 | 0 | 40.36 | 0 | 0 | 0 |

Figura 16.6 Una tabella trasposta.

In realtà questa trasposizione è stata effettuata anche nell'esempio di datazione delle fatture, dove DataFattura, divisa in periodi di 30 giorni, è stata convertita in quattro colonne.

Esiste un approccio anche più complicato, che consiste nel costruire dinamicamente una vista da una query dei valori dei nomi dei clienti, creando nuove colonne ogni volta che vengono aggiunti nuovi clienti. Questo richiede la rimozione e la nuova creazione della vista ogni volta che vengono aggiunti nuovi clienti, ma create view e select possono essere eseguite automaticamente, utilizzando le tecniche di generazione del codice descritte nel Capitolo 20.

16.4 Utilizzo di MOD in DECODE

La funzione modulo, MOD, può essere utilizzata insieme con DECODE e RowNum per produrre alcune stampe di avvertimento, gestione di documenti e altri effetti. RowNum è un'altra pseudocolonna e corrisponde al numero della riga contata mentre viene recuperata dal database; ovvero, quando si esegue un'istruzione select, alla prima riga viene assegnato un RowNum di 1, alla seconda di 2 e così via.

RowNum non è compresa nella locazione della riga nella tabella o database e non ha nessuna relazione con RowID. È un numero aggiunto alla riga quando viene prelevata dal database.

Nella Figura 16.7 un'istruzione select preleva DataFattura e Importo dalla tabella FATTURA. Nella colonna più a sinistra viene visualizzata RowNum. La seconda colonna è una combinazione di DECODE e MOD. Ogni valore di RowNum viene diviso per cinque nella funzione modulo. Il risultato corrisponde a *valore* in DECODE. Se RowNum è perfettamente divisibile per 5, il risultato di MOD è 0 e quindi il risultato di DECODE è RowNum. Se RowNum non è multiplo di cinque, il risultato di DECODE non è 0, perciò DECODE produce come risultato NULL.

Nella Figura 16.8 viene utilizzato lo stesso metodo di base, ma la colonna RowNum non viene più stampata e DECODE diventa una colonna distinta dopo Importo. Inoltre, ogni cinque righe invece di RowNum viene visualizzato un carattere di inizio pagina, inserito come *then* in DECODE. La colonna DECODE è stata rinominata A e un comando column ha fatto in modo che sia lunga solo un carattere, senza intestazione.

```
column Line Format 9999
select RowNum,DECODE(MOD(RowNum,5),0,RowNum, NULL) Line,
       DataFattura,Importo
  from FATTURA;
```

| ROWNUM | LINE | Data | Fattura | IMPORTO |
|--------|------|-----------|---------|---------|
| 1 | | 23-OCT-01 | 5.03 | |
| 2 | | 09-NOV-01 | 2.02 | |
| 3 | | 12-SEP-01 | 11.12 | |
| 4 | | 09-NOV-01 | 22.10 | |
| 5 | 5 | 17-NOV-01 | 8.29 | |
| 6 | | 01-SEP-01 | 21.32 | |
| 7 | | 15-NOV-01 | 7.33 | |
| 8 | | 04-OCT-01 | 8.42 | |
| 9 | | 04-OCT-01 | 1.43 | |
| 10 | 10 | 13-OCT-01 | 12.41 | |
| 11 | | 23-OCT-01 | 4.49 | |
| 12 | | 23-NOV-01 | 40.36 | |
| 13 | | 30-OCT-01 | 7.47 | |
| 14 | | 03-OCT-01 | 3.55 | |
| 15 | 15 | 22-OCT-01 | 13.65 | |
| 16 | | 21-SEP-01 | 9.87 | |
| 17 | | 22-OCT-01 | 17.58 | |
| 18 | | 09-OCT-01 | 8.98 | |

Figura 16.7 Numerazione delle righe utilizzando DECODE e MOD.

```

column A format al heading ''
select DataFattura, Importo,
       DECODE(MOD(RowNum,5),0,'1',null) A
from FATTURA;

Data
Fattura    IMPORTO
-----
23-OCT-01   5.03
09-NOV-01   2.02
12-SEP-01   11.12
09-NOV-01   22.10
17-NOV-01   8.29 ?
01-SEP-01   21.32
15-NOV-01   7.33
04-OCT-01   8.42
04-OCT-01   1.43
13-OCT-01   12.41 ?
23-OCT-01   4.49
23-NOV-01   40.36
30-OCT-01   7.47
03-OCT-01   3.55
22-OCT-01   13.65 ?
21-SEP-01   9.87
22-OCT-01   17.58
09-OCT-01   8.98

```

Figura 16.8 Elenco con inseriti dei caratteri di fine modulo.

In stampa si ottiene una nuova pagina di dati ogni volta che la stampante riceve il carattere di inizio modulo, come mostrato nella Figura 16.9. Le colonne vengono allineate correttamente durante la stampa. Se il sistema del computer non consente di digitare un valore esadecimale (in base 16) direttamente in una riga di testo, occorre utilizzare al posto del letterale l'espressione SQL CHR(12).

Un carattere esadecimale non è la sola opzione. Si possono utilizzare due caratteri di fine riga o una parola circondata da caratteri esadecimali, e così via. L'effetto di questa tecnica è quello di fornire una logica procedurale riga per riga. Se si comprende questo, si hanno grandi possibilità di utilizzo, ad esempio per confermare grandi blocchi di insert, update o delete sostituendo la parola commit al carattere esadecimale dell'ultimo esempio.

16.5 order by e RowNum

Poiché RowNum può essere utilizzato in maniera così vantaggiosa in DECODE, è importante tenere presente che questo numero è assegnato a una riga esattamente

| Data Fattura | IMPORTO |
|--------------|---------|
| 23-OCT-01 | 5.03 |
| 09-NOV-01 | 2.02 |
| 12-SEP-01 | 11.12 |
| 09-NOV-01 | 22.10 |
| 17-NOV-01 | 8.29 |

| | |
|-----------|-------|
| 01-SEP-01 | 21.32 |
| 15-NOV-01 | 7.33 |
| 04-OCT-01 | 8.42 |
| 04-OCT-01 | 1.43 |
| 13-OCT-01 | 12.41 |

| | |
|-----------|-------|
| 23-OCT-01 | 4.49 |
| 23-NOV-01 | 40.36 |
| 30-OCT-01 | 7.47 |
| 03-OCT-01 | 3.55 |
| 22-OCT-01 | 13.65 |

| | |
|-----------|-------|
| 21-SEP-01 | 9.87 |
| 22-OCT-01 | 17.58 |
| 09-OCT-01 | 8.98 |

Figura 16.9 Effetto della stampa della tabella mostrata nella Figura 16.8, quattro pagine distinte.

quando questa viene prelevata dal database, prima che ORACLE esegua qualsiasi istruzione order by. Come risultato, se si prova ad aggiungere un order by alla prima query che utilizzava DECODE e MOD, si ottengono i risultati che sono illustrati nella Figura 16.10.

Si può vedere che order by ha riaggiustato l'ordine delle righe, vanificando l'utilità di DECODE.

NOTA Se è importante disporre le righe in un determinato ordine, ad esempio in base a DataFattura, e anche utilizzare le caratteristiche della combinazione di DECODE, MOD e RowNum, si può creare una vista dove l'ordinamento è effettuato da group by. Per informazioni riguardo a questa pratica si può consultare il paragrafo “order by nelle viste”.

```
create or replace view DATAEIMPORTO as
select DataFattura, Importo
  from FATTURA
 group by DataFattura, Importo;
```

```
select RowNum,DECODE(MOD(RowNum,5),0,RowNum,NULL) Line,
       DataFattura,Importo
  from FATTURA;
 order by DataFattura;
```

| ROWNUM | LINE | Data Fattura | IMPORTO |
|--------|------|-----------------|---------|
| 6 | | 01-SEP-01 | 21.32 |
| 3 | | 12-SEP-01 | 11.12 |
| 16 | | 21-SEP-01 | 9.87 |
| 14 | | 03-OCT-01 | 3.55 |
| 8 | | 04-OCT-01 | 8.42 |
| 9 | | 04-OCT-01 | 1.43 |
| 18 | | 09-OCT-01 | 8.98 |
| 10 | 10 | 13-OCT-01 | 12.41 |
| 15 | 15 | 22-OCT-01 | 13.65 |
| 17 | | 22-OCT-01 | 17.58 |
| 1 | | 23-OCT-01 | 5.03 |
| 11 | | 23-OCT-01 | 4.49 |
| 13 | | 30-OCT-01 | 7.47 |
| 2 | | 09-NOV-01 | 2.02 |
| 4 | | 09-NOV-01 | 22.10 |
| 7 | | 15-NOV-01 | 7.33 |
| 5 | 5 | 17-NOV-01 | 8.29 |
| 12 | | 23-NOV-01 | 40.36 |

Figura 16.10 L'effetto di order by su RowNum.

L'istruzione select assegna poi a ogni riga il numero come viene restituito da questa vista, per cui l'obiettivo di ordinare le date e utilizzare DECODE, MOD e RowNum è realizzato, come mostrato nella Figura 16.11.

16.6 Colonne e calcoli in then ed else

Finora i nomi di colonna e i calcoli sono stati posti solo nella parte *valore* di DECODE, ma possono trovarsi facilmente anche in *then* e *else*. Si supponga che Talbot abbia una tabella PAGA che elenca i lavoratori le loro retribuzioni giornaliere, come mostrato nella Figura 16.12.

Talbot ha anche creato una vista, che calcola la retribuzione media di tutti i suoi lavoratori:

```
create or replace view PAGAMEDIA as
select AVG(PagaGiorno) PagaMedia
  from PAGA;
```

```
select RowNum,DECODE(MOD(RowNum,5),0,RowNum,NULL) Line,
       DataFattura,Importo
  from DATAEIMPORTO;
```

| ROWNUM | LINE | Data Fattura | IMPORTO |
|--------|------|-----------------|---------|
| 1 | | 01-SEP-01 | 21.32 |
| 2 | | 12-SEP-01 | 11.12 |
| 3 | | 21-SEP-01 | 9.87 |
| 4 | | 03-OCT-01 | 3.55 |
| 5 | 5 | 04-OCT-01 | 1.43 |
| 6 | | 04-OCT-01 | 8.42 |
| 7 | | 09-OCT-01 | 8.98 |
| 8 | | 13-OCT-01 | 12.41 |
| 9 | | 22-OCT-01 | 13.65 |
| 10 | 10 | 22-OCT-01 | 17.58 |
| 11 | | 23-OCT-01 | 4.49 |
| 12 | | 23-OCT-01 | 5.03 |
| 13 | | 30-OCT-01 | 7.47 |
| 14 | | 09-NOV-01 | 2.02 |
| 15 | 15 | 09-NOV-01 | 22.10 |
| 16 | | 15-NOV-01 | 7.33 |
| 17 | | 17-NOV-01 | 8.29 |
| 18 | | 23-NOV-01 | 40.36 |

Figura 16.11 Utilizzo di group by invece di order by per RowNum.

```
column PagaGiorno format 9999999.99
select Nome, PagaGiorno from PAGA;
```

| NOME | PAGAGIORNO |
|---------------|------------|
| ADAH TALBOT | 1.00 |
| ANDREW DYE | .75 |
| BART SARJEANT | .75 |
| DICK JONES | 1.00 |
| GEORGE OSCAR | 1.25 |
| PAT LAVAY | 1.25 |

Figura 16.12 I lavoratori di Talbot e l'ammontare della retribuzione giornaliera.

Poi Talbot decide di concedere ai suoi lavoratori un aumento di stipendio. I lavoratori che guadagnano esattamente la paga media continueranno a ricevere la stessa paga, gli altri otterranno un aumento salariale del 15 per cento. In questa istruzione select, PagaGiorno corrisponde a *valore* in DECODE, PagaMedia a *if*, ancora PagaGiorno a *then* e PagaGiorno moltiplicata per 1.15 a *else*. In questo modo si può vedere l'utilizzo di colonne e calcoli nelle parti *if*, *then* e *else* di DECODE, come illustrato nella Figura 16.13.

```
column PagaNuova format 9999999.9

select Nome, PagaGiorno,
       DECODE(PagaGiorno, PagaMedia, PagaGiorno,
               PagaGiorno*1.15) PagaNuova
  from PAGA, PAGAMEDIA;
```

| NOME | PAGAGIORNO | PAGANUOVA |
|---------------|------------|-----------|
| ADAH TALBOT | 1 | 1.00 |
| ANDREW DYE | .75 | .86 |
| BART SARJEANT | .75 | .86 |
| DICK JONES | 1 | 1.00 |
| GEORGE OSCAR | 1.25 | 1.44 |
| PAT LAVAY | 1.25 | 1.44 |

Figura 16.13 Un aumento salariale del 15 per cento per i lavoratori la cui retribuzione non è pari a quella media.

16.7 Maggiore, minore e uguale in DECODE

La sintassi di DECODE potrebbe indurre a credere che in realtà sia possibile effettuare solo una serie di confronti di uguaglianza, dal momento che viene controllata l'uguaglianza di *valore* con un elenco di *if*. Tuttavia, un ingegnoso utilizzo di funzioni e calcoli al posto di *valore* può concedere a DECODE la capacità di intraprendere azioni basate sul fatto che un valore sia maggiore, minore o uguale a un altro.

Nella Figura 16.14, ad esempio, PagaGiorno viene confrontato con PagaMedia per ogni lavoratore. A quelli che guadagnano più di PagaMedia viene concesso un aumento del 5 per cento, a quelli che guadagnano meno un aumento del 15 per cento. Quelli che guadagnano esattamente la retribuzione media non ottengono alcun aumento. Come si realizza questo?

PagaGiorno viene diviso per PagaMedia. Per quelli che guadagnano più della paga media il rapporto è un numero maggiore di 1, per quelli che guadagnano esattamente la retribuzione media il rapporto è 1, per quelli che guadagnano meno della retribuzione media il rapporto è un numero minore di uno e maggiore di 0.

Ora si deve sottrarre 1 da ognuno di questi numeri, che diventeranno rispettivamente un numero maggiore di zero, zero e un numero minore di zero.

```
select Nome, PagaGiorno,
       DECODE( SIGN( (PagaGiorno/PagaMedia)-1 ),
              1, PagaGiorno*1.05, -1, PagaGiorno*1.15, PagaGiorno) PagaNuova
  from PAGA, PAGAMEDIA;
```

| NOME | PAGAGIORNO | PAGANUOVA |
|---------------|------------|-----------|
| ADAH TALBOT | 1.00 | 1.00 |
| ANDREW DYE | .75 | .86 |
| BART SARJEANT | .75 | .86 |
| DICK JONES | 1.00 | 1.00 |
| GEORGE OSCAR | 1.25 | 1.31 |
| PAT LAVAY | 1.25 | 1.31 |

Figura 16.14 Aumenti salariali basati sul rapporto fra la retribuzione del lavoratore e quella media.

Applicando `SIGN` a ciascuno di essi si ottiene 1, 0 e -1 (per una trattazione di `SIGN` occorre consultare il Capitolo 7). Perciò *valore* in `DECODE` è 1, 0 o -1, a seconda della retribuzione. Se è 1, `PagaGiorno` viene moltiplicato per 1.05; se è -1, `PagaGiorno` viene moltiplicato per 1.15; se è 0 (*else*), il lavoratore continua a guadagnare `PagaGiorno`.

Questo metodo può essere utilizzato per realizzare aggiornamenti complessi con un solo passaggio, mentre l'alternativa sarebbe quella di impiegare una serie di istruzioni `update` con differenti istruzioni `set` e clausole `where`.

Lo stesso approccio può essere impiegato anche con altre funzioni. `GREATEST`, con un elenco di colonne, potrebbe corrispondere a *valore* in `DECODE` oppure a *if, then o else*. Quando serve si può ottenere una logica procedurale estremamente complessa.

Per riassumere questo capitolo, `DECODE` è una potente funzione che utilizza la logica *if, then, else* e che può essere combinata con altre funzioni di ORACLE per produrre distribuzioni di valori (come nella datazione delle fatture), per trasporre tabelle, per controllare la visualizzazione e gli spostamenti di pagina, per effettuare calcoli e per indurre cambiamenti riga per riga in base al valore di una colonna (o a un calcolo).

• Capitolo 17

• **Creazione, scaricamento e modifica di tabelle e viste**

- 17.1 **Creazione di una tabella**
- 17.2 **Scaricamento di tabelle**
- 17.3 **Modifica di tabelle**
- 17.4 **Creazione di una vista**
- 17.5 **Creazione di una tabella a partire da un'altra**
- 17.6 **Creazione di una tabella di solo indice**
- 17.7 **Utilizzo di tabelle partizionate**

Finora si è posta particolare attenzione all'utilizzo delle tabelle; in questo capitolo viene mostrato come creare, scaricare e modificare tabelle, come creare viste e come creare una tabella partendo da un'altra.

17.1 **Creazione di una tabella**

Si consideri la tabella STRANO; è simile alla tabella COMFORT, ma è utilizzata per registrare le città con valori di temperatura insoliti.

```
describe STRANO
```

| Name | Null? | Type |
|----------------|----------|--------------|
| CITTA | NOT NULL | VARCHAR2(13) |
| DATAAMPIONE | NOT NULL | DATE |
| MEZZOGIORNO | | NUMBER(3,1) |
| MEZZANOTTE | | NUMBER(3,1) |
| PRECIPITAZIONE | | NUMBER |

Le cinque righe della tabella STRANO rappresentano i tre principali tipi di dati in ORACLE, VARCHAR2, DATE e NUMBER, e a ogni colonna è stato assegnato uno di questi tipi quando la tabella è stata creata con la seguente istruzione SQL:

```
create table STRANO (Citta VARCHAR2(13) NOT NULL, DataCampione DATE NOT NULL, Mezzogiorno NUMBER(3,1), Mezzanotte NUMBER(3,1), Precipitazione NUMBER);
```

Un altro stile per scrivere lo stesso comando, meno incline a errori nella disposizione e nel conteggio delle parentesi, è il seguente:

```
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 virgolette. 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 (per ulteriori informazioni si consulti il paragrafo “Nomi di oggetto” nella guida alfabetica di riferimento del Capitolo 37).

Nella creazione delle tabelle non vi è distinzione fra lettere maiuscole e minuscole. Non vi sono opzioni per il tipo di dati DATE. Per i tipi di dati carattere deve essere specificata la lunghezza massima. I NUMBER possono avere precisione elevata (fino a 38 caratteri) o precisione fissata in base al massimo numero di cifre o al numero di posizioni consentite a destra del punto decimale (un campo Importo per la valuta americana, ad esempio, dovrebbe avere solo due posizioni decimali).

NOTA *Non si devono racchiudere i nomi di tabelle e colonne tra doppi apici, altrimenti le maiuscole e le minuscole avrebbero rilevanza. Questo potrebbe infatti creare gravi problemi per gli utenti o gli sviluppatori.*

Per altre opzioni di `create table` riguardo caratteristiche che si riferiscono agli oggetti, disponibili a partire da ORACLE8, si rimanda ai Capitoli 25, 26 e 31.

Larghezza delle colonne di caratteri e precisione dei NUMBER

Specificare una lunghezza massima per le colonne di caratteri (CHAR e VARCHAR2) e una precisione per le colonne NUMBER può avere conseguenze che devono essere considerate durante il progetto di una tabella. Decisioni sbagliate possono essere corrette in seguito utilizzando il comando `alter table`, ma con difficoltà.

La larghezza adeguata Se una colonna di caratteri non è larga abbastanza per i dati da inserirvi, l'istruzione insert fallisce, producendo come risultato questo messaggio di errore:

```
ERROR at line 1: ORA-1401: inserted value too large for column
```

La larghezza massima per una colonna di CHAR (valore fisso) è di 2000 caratteri (in ORACLE 7 era limitata a 255 caratteri). Una colonna di VARCHAR2 può avere fino a 4000 caratteri (in ORACLE 7 erano limitati a 2000). Quando si assegna la larghezza a una colonna, occorre riservare abbastanza spazio per tutte le reali possibilità future. Ad esempio, una colonna CHAR(15) per il nome di una città può creare problemi in seguito, per cui si dovrà modificare la tabella oppure troncare il nomi di alcune città.

NOTA *Non vi è alcuno svantaggio nel definire in ORACLE una grande colonna di VARCHAR2, infatti gli spazi vuoti alla fine di queste colonne non vengono salvati. Il nome della città "SAN FRANCISCO", ad esempio, viene salvato in 13 spazi anche se la colonna è stata definita come VARCHAR2(50) e non vengono salvati 37 spazi vuoti con il nome. La colonna salvata ha soltanto i caratteri effettivi. Se in una colonna non vi è nulla (NULL), ORACLE non vi salva niente, nemmeno uno spazio vuoto (in realtà salva una coppia 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 di default di SQLPLUS per le colonne. Infatti SQLPLUS crea un'intestazione di default di lunghezza 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 devono essere inserite in ORACLE (le temperature in questo caso sono espresse in gradi Fahrenheit):

```
insert into STRANO values
('PLEASANT LAKE', TO_DATE('21-MAR-1993','DD-MON-YYYY'),
 39.99, -1.31, 3.6);

insert into STRANO values
('PLEASANT LAKE', TO_DATE('22-JUN-1993','DD-MON-YYYY'),
 101.44, 86.2, 1.63);

insert into STRANO values
('PLEASANT LAKE', TO_DATE('23-SEP-1993','DD-MON-YYYY'),
 92.85, 79.6, 1.00003);

insert into STRANO values
('PLEASANT LAKE', TO_DATE('22-DEC-1993','DD-MON-YYYY'),
 -17.445, -10.4, 2.4);
```

Ed ecco i risultati:

```
1 row created.
```

```
insert into STRANO values ('PLEASANT LAKE', TO_DATE('22-JUN-1993','DD-MON-  
YYYY'), 101.44, 86.2, 1.63)
```

```
ERROR at line 1: ORA-1438: value larger than specified precision allows  
for this column  
1 row created.
```

```
1 row created.
```

La prima, la terza e la quarta riga sono state inserite, ma 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 massimo numero di cifre che vengono salvate, 1 specifica che una di queste cifre è riservata a una posizione a destra del punto decimale. Perciò, 12.3 è un numero lecito, ma 123.4 no.

Si osservi che qui 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), ma viene semplicemente arrotondato a una sola cifra decimale. Questo fatto verrà illustrato tra breve, ma prima occorre esaminare i risultati di una query delle quattro colonne che si è cercato di inserire (la larghezza è stata adattata per mostrare l'intera intestazione):

```
select * from STRANO;
```

| CITTA | DATACAMPIONE | MEZZOGIORNO | MEZZANOTTE | PRECIPITAZIONE |
|---------------|--------------|-------------|------------|----------------|
| PLEASANT LAKE | 21-MAR-93 | 39.9 | -1.31 | 3.6 |
| PLEASANT LAKE | 23-SEP-93 | 92.9 | 79.6 | 1.00003 |
| PLEASANT LAKE | 22-DEC-93 | -17.4 | -10.4 | 2.4 |

Tre righe sono state inserite correttamente; manca solo quella che ha creato problemi. ORACLE ha annullato automaticamente l'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 qui:

```
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:

```
select * from STRANO;
```

| CITTA | DATA | CAMPIONE MEZZOGIORNO | MEZZANOTTE | PRECIPITAZIONE |
|---------------|-----------|----------------------|------------|----------------|
| PLEASANT LAKE | 21-MAR-93 | 40 | -1.3 | 3.6 |
| PLEASANT LAKE | 22-JUN-93 | 101.4 | 86.2 | 1.63 |
| PLEASANT LAKE | 23-SEP-93 | 92.9 | 79.6 | 1.00003 |
| PLEASANT LAKE | 22-DEC-93 | -17.4 | -10.4 | 2.4 |

Si esamini la prima istruzione insert. Il valore per Mezzogiorno è 39.99 e nella query è arrotondato a 40. Mezzanotte nell'istruzione insert vale -1.31 nella query vale -1.3. ORACLE arrotonda il numero in base alla cifra immediatamente a destra della precisione consentita. La Tabella 17.1 mostra gli effetti della precisione in vari esempi.

Tabella 17.1 Esempi degli effetti della precisione sui valori inseriti. (*continua*)

| NELL'ISTRUZIONE INSERT | VALORE REALE NELLA TABELLA |
|-----------------------------|----------------------------|
| Con precisione NUMBER(4,1) | |
| 123.4 | 123.4 |
| 123.44 | 123.4 |
| 123.45 | 123.5 |
| 123.445 | 123.4 |
| 1234.5 | insert fallisce |
| Con precisione NUMBER(4) | |
| 123.4 | 123 |
| 123.44 | 123 |
| 123.45 | 123 |
| 123.445 | 123 |
| 1234.5 | 1235 |
| 12345 | insert fallisce |
| Con precisione NUMBER(4,-1) | |
| 123.4 | 120 |
| 123.44 | 120 |
| 123.45 | 120 |
| 123.445 | 120 |
| 125 | 130 |
| 1234.5 | 1230 |
| 12345 | insert fallisce |

Tabella 17.1 Esempi degli effetti della precisione sui valori inseriti.

| NELL'ISTRUZIONE INSERT | VALORE REALE NELLA TABELLA |
|------------------------|----------------------------|
| Con precisione NUMBER | |
| 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 |

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 quello di fare in modo che ORACLE effettui la maggior parte del lavoro per conservare l'integrità della tabella. Più sono i vincoli aggiunti a una definizione di tabella, minore è il lavoro da svolgere nell'applicazione per la gestione dei dati. D'altra parte, più vincoli vi sono in una tabella, maggiore è il tempo necessario per aggiornarla.

Esistono due modi per specificare dei 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 varie 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 univocamente ciascuna riga di una tabella.

```
create table STRANO (
    Citta          VARCHAR2(13) NOT NULL,
    DataCampione   DATE NOT NULL,
    Mezzogiorno    NUMBER(4,1),
    Mezzanotte     NUMBER(4,1),
    Precipitazione NUMBER,
    UNIQUE (Citta, DataCampione)
);
```

La chiave di questa tabella è una combinazione di `Citta` e `DataCampione`. Si noti che entrambe le colonne sono dichiarate anche come `NOT NULL`. Questa caratteristica consente di evitare che vengano inserite nella tabella righe in cui determinate colonne non abbiano alcun valore. 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`.

La chiave primaria La chiave primaria di una tabella è una delle chiavi candidate a cui vengono assegnate determinate caratteristiche. Può esistere una sola chiave primaria e una colonna di tale chiave non può contenere valori nulli.

```
create table STRANO (
    Citta          VARCHAR2(13),
    DataCampione   DATE,
    Mezzogiorno    NUMBER(4,1),
    Mezzanotte     NUMBER(4,1),
    Precipitazione NUMBER,
    PRIMARY KEY (Citta, DataCampione)
);
```

Questa istruzione `create table` ha lo stesso effetto di quella precedente, eccetto che si possono avere vari vincoli `UNIQUE` ma solo un vincolo `PRIMARY KEY`.

Per chiavi primarie o candidate di una sola colonna, si può definire la chiave con un vincolo di colonna invece di un vincolo di tabella

```
create table LAVORATORE (
    Nome          VARCHAR2(25) PRIMARY KEY,
    Eta           NUMBER,
    Alloggio      VARCHAR2(15)
);
```

In questo caso la colonna `Nome` è la chiave primaria.

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 `LAVORATORE`, ad esempio, la colonna `Alloggio` fa riferimento ai valori della colonna `Alloggio` nella tabella `ALLOGGIO`:

```
create table LAVORATORE (
    Nome          VARCHAR2(25) PRIMARY KEY,
    Eta           NUMBER,
    Alloggio      VARCHAR2(15) REFERENCES ALLOGGIO(Alloggio)
);
```

Nella clausola `references` si può fare riferimento a una chiave primaria o unica anche nella stessa tabella, ma non a una tabella in un database remoto. Per specificare chiavi esterne con più colonne si utilizza la forma per le tabelle (che è stata impiegata nell'esempio precedente per creare una `PRIMARY KEY` sulla tabella `STRANO`) invece della forma per le colonne.

A volte si desidera cancellare le righe dipendenti quando si elimina la riga da cui dipendono. Nel caso di `LAVORATORE` e `ALLOGGIO`, quando si cancella `ALLOGGIO`, si vuole rendere nulla la colonna `Alloggio`. 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 da cui dipende. Per maggiori informazioni sulle clausole `on delete cascade` e `references` si può consultare il paragrafo dedicato ai vincoli di integrità nella guida alfabetica di riferimento (Capitolo 37).

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 fornire un'espressione che deve essere sempre vera per ogni riga della tabella. Ad esempio, se Talbot assume solo lavoratori con età compresa fra i 18 e i 65 anni, il vincolo dovrebbe essere simile a questo:

```
create table LAVORATORE (
  Nome      CHAR(25) PRIMARY KEY,
  Eta       NUMBER CHECK (Eta BETWEEN 18 AND 65),
  Alloggio   CHAR(15) REFERENCES ALLOGGIO(Alloggio)
);
```

Non si può fare riferimento a valori in altre righe e non si possono denominare le pseudocolonne SysDate, UID, User, Userenv, Curval, Nextval, Level o RowNum. Si può utilizzare la forma di vincolo delle tabelle (invece di quella per le colonne) per fare riferimento a più colonne nel vincolo di controllo.

Assegnazione di nomi ai vincoli

È possibile assegnare dei nomi ai vincoli. Se si ha un efficiente schema di designazione per i nomi dei vincoli, si è in grado di identificarli e gestirli in modo migliore. Il nome di un vincolo dovrebbe identificare la tabella su cui agisce e il tipo di limite. Ad esempio, la chiave primaria sulla tabella STRANO si potrebbe chiamare STRANO_PK.

Si può specificare un nome per un limite quando viene creato, altrimenti ne viene generato uno da ORACLE. La maggior parte dei nomi di vincoli generati da ORACLE è nella forma SYS_C#####; ad esempio SYS_C000145. Poiché i nomi dei vincoli generati dal sistema non offrono alcuna indicazione sulla tabella o sul vincolo, è opportuno assegnare sempre dei nomi ai vincoli, in particolare ai vincoli PRIMARY KEY, FOREIGN KEY, CHECK e UNIQUE.

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 limite (in questo caso STRANO_PK). Si può utilizzare questo nome in seguito, quando si attiva o disattiva il vincolo.

17.2 Scaricamento di tabelle

Scaricare 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 scaricate quando non servono più. In ORACLE il comando truncate consente di rimuovere tutte le righe di una tabella e recuperare lo spazio per altri utenti senza rimuovere le definizioni della tabella dal database.

Anche il troncamento è molto semplice.

```
truncate table STRANO;
```

Table truncated.

Il troncamento non può essere annullato. Se vi sono trigger che cancellano righe che dipendono da righe della tabella, questi non vengono eseguiti dal troncamento. Occorre essere sicuri di voler davvero effettuare il troncamento, prima di procedere con l'operazione.

17.3 Modifica di tabelle

Esistono due modi per modificare le tabelle: cambiando una definizione di colonna o aggiungendo una colonna alla tabella esistente. Aggiungere una colonna è semplice e simile a creare una tabella. Si supponga che si decida di 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 è il seguente:

```
alter table STRANO add (
Condizione    VARCHAR2(9) NOT NULL,
Vento         NUMBER(3)
);
```

```
alter table STRANO add (
*
```

ERROR at line 1: ORA-1758: 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 quando si cerca di inserirla è vuota e quindi ogni riga nella tabella avrebbe una nuova colonna vuota definita come NOT NULL.

Esistono due alternative. La clausola add del comando alter table funziona con una colonna NOT NULL se la tabella è vuota. Tuttavia, di solito 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.

Poi si deve riempire la colonna con dei dati per ogni riga (dati validi oppure finti, finché non si ottengono quelli validi):

```
update STRANO set Condizione = 'SOLE';
```

Infine occorre modificare di nuovo la tabella con modify e cambiare la definizione della colonna come NOT NULL:

```
alter table STRANO modify (
Condizione    VARCHAR2(9) NOT NULL,
Citta         VARCHAR2(17)
);
```

Table altered.

Si noti che è stata modificata anche la colonna Citta, ampliata a 17 caratteri (solo per mostrare come eseguire questa modifica). Quando la tabella viene descritta, appare così:

```
describe STRANO
```

| Name | Null? | Type |
|---------------|----------|--------------|
| CITTA | NOT NULL | VARCHAR2(17) |
| DATAAMPIONE | NOT NULL | DATE |
| MEZZOGIORNO | | NUMBER(4,1) |
| MEZZANOTTE | | NUMBER(4,1) |
| PRECIPITAZIOE | | NUMBER |
| CONDIZIONE | NOT NULL | VARCHAR2(9) |
| VENTO | | NUMBER(3) |

E contiene quanto segue:

```
select * from STRANO;
```

| CITTA | DATAAMPIONE | MEZZOGIORNO | MEZZANOTTE | PRECIP | CONDIZIONE | VENTO |
|---------------|-------------|-------------|------------|---------|------------|-------|
| PLEASANT LAKE | 21-MAR-93 | 40 | -1.3 | 3.6 | SOLE | |
| PLEASANT LAKE | 22-JUN-93 | 101.4 | 86.2 | 1.63 | SOLE | |
| PLEASANT LAKE | 23-SEP-93 | 92.9 | 79.6 | 1.00003 | SOLE | |
| PLEASANT LAKE | 22-DEC-93 | -17.4 | -10.4 | 2.4 | SOLE | |

Qui si possono vedere gli effetti delle modifiche. Citta ora è ampia 17 caratteri invece che 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 modificare una colonna NOT NULL in modo che possa contenere anche valori nulli, si deve utilizzare il comando alter table con la clausola NULL, come mostrato nel seguente listato:

```
alter table STRANO modify  
(Condizione NULL);
```

Le 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.

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, possono essere effettuate le seguenti modifiche.

- Si può cambiare il suo tipo di dati.
- Si può diminuire la larghezza di una colonna di caratteri.
- Si può diminuire il numero di cifre in una colonna NUMBER.

Esistono altre opzioni disponibili in differenti versioni di ORACLE, descritte nella guida alfabetica di riferimento (Capitolo 37) sotto alter table.

17.4 Creazione di una vista

Dal momento che le tecniche per creare una vista sono state già presentate (nei Capitoli 3, 10, 12, 13 e 16), non vengono trattate di nuovo. Però in questo paragrafo sono spiegati altri concetti sulle viste che si dimostreranno utili.

Se una vista è basata su una singola tabella sottostante, è possibile inserire, aggiornare o cancellare righe nella vista stessa. In questo modo vengono effettivamente inserite, aggiornate o cancellate righe nella tabella sottostante. Vi sono alcune restrizioni, abbastanza ovvie, alla possibilità di fare questo.

- Non è possibile effettuare inserimenti se la tabella sottostante ha una colonna NOT NULL che non compare nella vista.
- Non è possibile effettuare inserimenti o aggiornamenti se qualcuna delle colonne della vista a 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.

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, ma è semplicemente una descrizione (un'istruzione SQL) delle informazioni da prelevare da altre tabelle e di come organizzarle. Di conseguenza, se una tabella viene rimossa, la vista non ha più alcuna validità. Se si cerca di eseguire una query in una vista dove la tabella sottostante è stata rimossa, si ottiene un messaggio di errore riguardo alla vista. Nella sequenza che segue si crea una vista su una tabella esistente, si rimuove la tabella e poi si effettua una query della vista.

Innanzitutto viene creata la vista:

```
create view PIOGGIA as  
select Citta, Precipitazione  
from STRANO;
```

View created.

La tabella sottostante viene scaricata:

```
drop table STRANO;
```

Table dropped.

Infine viene effettuata la query della vista:

```
select * from PIOGGIA  
*  
ERROR at line 1: ORA-0942: table or view does not exist
```

Similmente, viene creata una vista utilizzando l'asterisco:

```
create view PIOGGIA as  
select * from STRANO;
```

View created.

e poi viene modifica la tabella sottostante:

```
alter table STRANO add (
Attenzione      VARCHAR2(20)
);
```

Table altered.

Nonostante la modifica alla tabella di base della vista, quest'ultima è ancora valida e accede alle nuove colonne della tabella STRANO.

Per creare una vista conservando tutti i privilegi che le sono stati concessi, occorre utilizzare il comando `create or replace view`, come mostrato nel seguente elenco. Questo comando sostituisce il testo di una vista esistente con quello della nuova vista, senza influire sulle vecchie concessioni.

```
create or replace view PIOGGIA as select * from STRANO;
```

order by nelle viste

Non è possibile utilizzare `order by` in un'istruzione `create view`. A volte si potrebbe realizzare lo stesso scopo con `group by`, che qui può essere impiegato, come si è mostrato nel Capitolo 6 al paragrafo "order by e RowNum". Tuttavia, anche se non vi sono funzioni di gruppo nella clausola `select`, la clausola `group by` può sempre consolidare le righe.

Si consideri questa query dalla tabella COMFORT con `order by`:

```
select Citta, Precipitazione
  from COMFORT
 order by Precipitazione;
```

| CITTA | PRECIPITAZIONE |
|---------------|----------------|
| KEENE | |
| SAN FRANCISCO | .1 |
| SAN FRANCISCO | .1 |
| SAN FRANCISCO | .5 |
| KEENE | 1.3 |
| SAN FRANCISCO | 2.3 |
| KEENE | 3.9 |
| KEENE | 4.4 |

8 rows selected.

La stessa query, utilizzando `group by` invece di `order by`, comprime le due righe identiche di SAN FRANCISCO (con Precipitazione di .1) in una sola:

```
create view DISCOMFORT as
select Citta, Precipitazione
  from COMFORT
 group by Precipitazione, Citta;
```

View created.

Quando viene eseguita la query, restano solo sette righe:

```
select * from DISCOMFORT;
```

| CITTA | PRECIPITAZIONE |
|---------------|----------------|
| KEENE | |
| SAN FRANCISCO | .1 |
| SAN FRANCISCO | .5 |
| KEENE | 1.3 |
| SAN FRANCISCO | 2.3 |
| KEENE | 3.9 |
| KEENE | 4.4 |

7 rows selected.

Questo probabilmente non è il risultato atteso. Anche se group by può ordinare una vista, utilizzarlo per questo scopo può causare dei problemi.

In genere è meglio utilizzare semplicemente order by nell'istruzione select che effettua la query nella vista.

Creazione di una vista di sola lettura

Si può 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 ha la capacità 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 svolgendo queste operazioni sulla vista PIOGGIA. Se la vista è basata sull'unione 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 da aggiornare non sia inclusa nelle colonne della vista.

Per evitare modifiche alle tabelle di base attraverso una vista, si può utilizzare la clausola with read only del comando create view. Se si impiega questa clausola quando si crea una vista, gli utenti possono soltanto selezionare record dalla vista, ma non manipolare i record ottenuti dalla vista stessa, anche se questa è basata su una sola tabella:

```
create or replace view PIOGGIA as
select * from STRANO
with read only;
```

Si può anche creare un trigger INSTEAD OF per gestire i comandi di manipolazione dei dati eseguiti sulla vista. Per una trattazione dettagliata dei trigger INSTEAD OF si rimanda al Capitolo 25.

17.5 Creazione di una tabella a partire da un'altra

La vista PIOGGIA creata precedentemente dalla tabella STRANO avrebbe potuto essere anche una tabella. ORACLE consente di creare in tempo reale una nuova tabella, basata su un'istruzione select in una tabella esistente:

```
create table PIOGGIA 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 caratteristiche 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 anche includere delle colonne “inventate”, che sono prodotti di funzioni o combinazioni 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 hanno una specifica precisione nella tabella di origine, ma sono sottoposte a calcoli nella creazione di una nuova colonna, sono semplicemente colonne NUMBER, con una precisione non specificata, nella nuova tabella. Quando la tabella PIOGGIA viene descritta, mostra le definizioni delle colonne:

```
describe PIOGGIA
```

| Name | Null? | Type |
|----------------|----------|--------------|
| CITTA | NOT NULL | VARCHAR2(13) |
| PRECIPITAZIONE | | NUMBER |

Quando viene effettuata una query su PIOGGIA, si può vedere che questa contiene proprio le colonne e i dati selezionati dalla tabella STRANO come segue:

```
select * from PIOGGIA;
```

| CITTA | PRECIPITAZIONE |
|---------------|----------------|
| PLEASANT LAKE | 3.6 |
| PLEASANT LAKE | 1.63 |
| PLEASANT LAKE | 1.00003 |
| PLEASANT LAKE | 2.4 |

Si può anche utilizzare questa tecnica 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 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;  
  
no rows selected.
```

Si può creare una tabella su una query senza generare *annotazioni redo log* (registrazioni cronologiche delle azioni del database utilizzate durante i ripristini). Per evitare la generazione di queste annotazioni si utilizza la parola chiave `nologging` nel comando `create table` (`nologging` sostituisce la parola chiave `unrecoverable` introdotta in ORACLE7.2). Quando si evitano in questo modo le annotazioni redo log, le prestazioni del comando `create table` migliorano, dal momento che deve essere svolto meno lavoro; 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 annotazioni redo log), la tabella non viene ricreata se, dopo un guasto del database, si utilizzano questi file per recuperarlo. Perciò conviene effettuare un backup del database subito dopo l'utilizzo dell'opzione `nologging`, se si vuole essere in grado di ripristinare la nuova tabella.

Il seguente esempio 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 annotazioni redo log.

```
create table PIOGGIA  
unrecoverable  
as  
select * from STRANO;
```

Table created.

17.6 Creazione di una tabella di solo indice

Una *tabella di solo indice* conserva i dati ordinati secondo i valori delle colonne di chiave primaria e li memorizza come se l'intera tabella fosse registrata in un indice. Gli indici sono descritti più dettagliatamente nel Capitolo 19; in ORACLE gli indici servono per due scopi principali, descritti di seguito.

- Per imporre l'unicità. Quando viene creato un vincolo `PRIMARY KEY` o `UNIQUE`, ORACLE produce un indice per imporre l'unicità delle colonne incluse.

- Per migliorare le prestazioni. Quando una query può utilizzare un indice, le sue prestazioni migliorano considerevolmente. Per informazioni dettagliate sulle condizioni sotto le quali una query può utilizzare un indice si rimanda al Capitolo 36.

Una tabella di solo indice consente di memorizzare i dati dell'intera tabella in un indice. Un normale indice registra solo le colonne incluse.

Poiché i dati della tabella sono salvati come un indice, le righe non hanno RowID. Perciò non è possibile selezionare da una tabella di solo indice i valori di questa pseudocolonna. Inoltre, non è possibile creare altri indici sulla tabella; il solo indice valido è l'indice della chiave primaria.

Per creare una tabella di solo indice, si deve utilizzare la clausola organization index del comando create table, come mostrato nel seguente esempio:

```
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, si deve creare su di essa un vincolo PRIMARY KEY.

Poiché non si può includere nell'indice alcun'altra colonna della tabella STRANO, una tabella di solo indice è appropriata se si accede ai suoi dati attraverso le colonne Citta e DataCampione (nella clausola where delle query). Per ridurre al minimo la quantità di lavoro necessario per la gestione dell'indice, conviene utilizzare tabelle di solo indice solo se i dati sono molto statici. Se questi cambiano frequentemente o se si ha necessità di includere nell'indice altre colonne della tabella, conviene impiegare una tabella normale, con gli indici appropriati.

17.7 Utilizzo di tabelle partizionate

A partire da ORACLE8 è possibile suddividere le righe di una tabella in più tabelle. I metodi utilizzati per determinare quali righe sono salvate nelle varie tabelle sono specificate nel comando create table. La suddivisione dei dati di una tabella in più tabelle è chiamata *partizionamento* della tabella; la tabella è detta *partizionata* e le parti sono dette *partizioni*.

Il partizionamento è utile per tabelle molto grandi. Suddividendo le righe di una tabella più grande in varie tabelle più piccole, si realizzano diversi obiettivi importanti.

- Le prestazioni delle query nelle tabelle possono migliorare, dal momento che per risolvere le query può bastare effettuare la ricerca solo in una partizione (una parte della tabella) invece che nell'intera tabella.

- La tabella può essere più facile da gestire. Poiché i dati delle tabelle partizionate vengono salvati in varie tabelle più piccole, può essere più semplice caricare e cancellare dati nelle partizioni invece che nella tabella grande.
- Le operazioni di backup e ripristino possono essere svolte in modo migliore. Dal momento che le partizioni sono più piccole della tabella partizionata, si possono avere più possibilità per effettuare il backup e il ripristino delle partizioni che per una sola grande tabella.

L'ottimizzatore di ORACLE comprende che la tabella è stata partizionata; come viene mostrato più avanti, è anche possibile specificare la partizione da utilizzare nella clausola from della query.

Creazione di una tabella partizionata

Per creare una tabella partizionata occorre specificare l'intervallo di valori da utilizzare per le partizioni nel comando create table.

Si consideri la tabella LAVORATORE:

```
create table LAVORATORE (
    Name      VARCHAR2(25),
    Eta       NUMBER,
    Alloggio  VARCHAR2(15),
    constraint LAVORATORE_PK PRIMARY KEY(Name)
);
```

Se si intende registrare un grande numero di record nella tabella LAVORATORE, è preferibile dividere le righe in più tabelle.

Per partizionare i record della tabella, occorre utilizzare la clausola partition by range del comando create table. Gli intervalli stabiliscono i valori memorizzati in ogni partizione.

```
create table LAVORATORE (
    Name      VARCHAR2(25),
    Eta       NUMBER,
    Alloggio  VARCHAR2(15),
    constraint LAVORATORE_PK PRIMARY KEY(Name)
)
partition by range (Alloggio)
(partition PART1 values less than ('F')
  tablespace PART1_TS,
 partition PART2 values less than ('N')
  tablespace PART2_TS,
 partition PART3 values less than ('T')
  tablespace PART3_TS,
 partition PART4 values less than (MAXVALUE)
  tablespace PART4_TS)
;
```

La tabella LAVORATORE viene partizionata in base ai valori della colonna Alloggio:

```
partition by range (Alloggio)
```

Per qualsiasi valore di Alloggio minore di ‘F’, il record viene salvato nella partizione chiamata PART1. Questa partizione viene memorizzata nel tablespace PART1_TS (per maggiori dettagli sui tablespace si rimanda al Capitolo 19).

Qualsiasi Alloggio compreso nell’intervallo tra ‘F’ e ‘N’ viene salvato nella partizione PART2; valori tra ‘N’ e ‘T’ vengono salvati nella partizione PART3. Qualsiasi valore maggiore di ‘T’ viene salvato nella partizione PART4. Si noti che nella definizione di quest’ultima partizione la clausola dell’intervallo è la seguente:

```
partition PART4    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 salvare tutti i dati non inclusi nelle precedenti.

Si noti che, per ogni partizione, si specifica soltanto il valore massimo dell’intervallo; quello minimo viene determinato implicitamente da ORACLE.

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. Nel seguente listato viene mostrato il comando create index per la tabella LAVORATORE:

```
create index LAVORATORE_ALLOGGIO
on LAVORATORE(Alloggio)
local
(partition PART1
  tablespace PART1_NDX_TS,
partition PART2
  tablespace PART2_NDX_TS,
partition PART3
  tablespace PART3_NDX_TS,
partition PART4
  tablespace PART4_NDX_TS)
```

Si noti la parola chiave local. In questo comando create index non viene specificato alcun intervallo, invece la parola local comunica a ORACLE di creare un indice distinto per ogni partizione della tabella LAVORATORE. In LAVORATORE sono state create quattro partizioni, quindi questo indice crea quattro indici distinti, uno per ogni partizione. Dal momento che vi è un indice per partizione, gli indici sono “locali” alle partizioni.

Si possono creare anche indici “globali”. Un indice globale può contenere valori di più partizioni. Anche l’indice stesso può essere partizionato, come mostrato nel seguente esempio:

```

create index LAVORATORE_ALLOGGIO
on LAVORATORE(Alloggio)
global partition by range (Alloggio)
(partition PART1    values less than ('F')
  tablespace PART1_NDX_TS,
partition PART2    values less than ('N')
  tablespace PART2_NDX_TS,
partition PART3    values less than ('T')
  tablespace PART3_NDX_TS,
partition PART4    values less than (MAXVALUE)
  tablespace PART4_NDX_TS)
;

```

La clausola global in questo comando create index consente di specificare intervalli per i valori degli indici diversi dagli intervalli per le partizioni della tabella. Gli indici locali possono essere più facili da gestire di quelli globali, ma gli indici globali possono svolgere controlli di unicità più veloci di quelli locali (partizionati).

Gestione di tabelle partizionate

Si può utilizzare il comando alter table per aggiungere, rimuovere, permutare, spostare, modificare, rinominare, dividere e troncare partizioni. Questi comandi consentono di modificare la struttura delle partizioni esistenti, come potrebbe risultare necessario dopo che una tabella partizionata è stata intensamente utilizzata. Ad esempio, la distribuzione dei valori Alloggio in una tabella partizionata può essere cambiata, o il valore massimo può essere aumentato. Per maggiori informazioni su queste opzioni si può consultare la voce ALTER TABLE nella guida alfabetica di riferimento (Capitolo 37).

Realizzazione di query direttamente dalle partizioni

Se si conosce la partizione dove possono essere recuperati i dati, è possibile specificarne il nome nella clausola from della query. Ad esempio, come si fa a eseguire una query per ottenere i record dei lavoratori il cui alloggio comincia con la lettera ‘R’? L’ottimizzatore dovrebbe essere in grado di utilizzare le definizioni delle partizioni per stabilire che solo la partizione PART3 può contenere i dati per risolvere questa query. Se lo si desidera, si può indicare a ORACLE di utilizzare solo PART3 nella query.

```

select *
  from LAVORATORE (PART3)
 where Nome like 'R%';

```

Questo esempio nomina esplicitamente la partizione in cui ORACLE deve cercare i record di abitazioni che cominciano con ‘R’, perciò occorre prestare molta attenzione quando si impiega questa sintassi.

In generale questa sintassi non è necessaria, dal momento che ORACLE pone vincoli CHECK su ogni partizione. Quando si effettua una query dalla tabella parti-

zionata, ORACLE utilizza i vincoli CHECK per stabilire quali partizioni dovrebbero essere coinvolte nella risoluzione della query. Questo processo può fare in modo che la ricerca per la query venga effettuata solo su un piccolo numero di righe, migliorando le prestazioni. Inoltre le partizioni possono essere memorizzate in table-space diversi (e quindi su dischi distinti), contribuendo a ridurre il potenziale conflitto per l'I/O del disco durante lo svolgimento della query.

Durante un inserimento in una tabella partizionata, ORACLE utilizza i vincoli CHECK delle partizioni per stabilire in quale partizione devono essere inseriti i record. Perciò si può utilizzare una tabella partizionata come se fosse una tabella singola e affidarsi a ORACLE per la gestione della separazione interna dei dati.

Capitolo 18

ORACLE e l'autorità

18.1 Utenti, ruoli e privilegi

18.2 Che cosa possono concedere gli utenti

18.3 Concessione di risorse limitate

L'informazione è vitale per il successo, ma quando è danneggiata o si trova in mani sbagliate, possono verificarsi gravi problemi. ORACLE fornisce estese caratteristiche di sicurezza 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 (rimanendo indipendente) a qualsiasi sicurezza già presente sul sistema del proprio computer. Per controllare l'accesso ai dati ORACLE utilizza i comandi `create user`, `create role` e `grant`.

18.1 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 consiste in una serie di privilegi (o nel tipo di accesso di cui ogni utente ha necessità a seconda della sua posizione e delle sue responsabilità). È possibile concedere particolari privilegi a dei ruoli e assegnare i ruoli agli utenti appropriati. Un utente può anche concedere direttamente privilegi ad altri utenti.

I privilegi di sistema del database consentono di eseguire particolari serie di comandi. Il privilegio `CREATE TABLE`, ad esempio, consente di creare tabelle. Il privilegio `GRANT ANY PRIVILEGE` permette di concedere qualsiasi privilegio di sistema. Per un elenco completo dei privilegi occorre consultare il paragrafo relativo nella guida alfabetica di riferimento del Capitolo 37.

I privilegi di oggetti del database offrono la capacità di svolgere alcune operazioni su vari oggetti. Il privilegio `DELETE`, ad esempio, consente di cancellare righe dalle tabelle e dalle viste.

Il privilegio `SELECT` consente di effettuare query con un'istruzione `select` su tabelle, viste, sequenze e snapshot. Per un elenco completo occorre consultare il paragrafo dedicato ai privilegi nella guida alfabetica di riferimento del Capitolo 37.

Creazione di un utente

Il sistema ORACLE viene fornito con due utenti già creati, SYSTEM e SYS. Per creare altri occorre aprire l'accesso dell'utente SYSTEM, che ha questo privilegio.

L'amministratore del sistema, quando installa ORACLE, crea innanzitutto un utente per se stesso.

Questa è la sintassi del comando `create user`:

```
create user utente identified {by password | externally};
```

Vi sono altri privilegi che possono essere impostati attraverso questo comando; per informazioni dettagliate occorre cercare il comando `create user` nella guida alfabetica di riferimento del Capitolo 37.

Per connettere nomi utente e password del sistema del proprio computer nella sicurezza di ORACLE, in modo che serva una sola identificazione per aprire un accesso, 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. Nel seguente esempio l'amministratore del sistema viene chiamato Dora:

```
create user Dora identified by avocado;
```

Ora esiste l'account di Dora ed è assicurato con una password.

Si può anche installare l'utente con particolari *tablespace* (spazi sul disco per le tabelle degli utenti, trattati nel capitolo seguente) e *quote* per l'utilizzo di spazi e risorse. Per informazioni su su *tablespace* e risorse si può consultare la guida alfabetica di riferimento (Capitolo 37) e il Capitolo 19.

Per cambiare la password occorre utilizzare il comando `alter user`:

```
alter user Dora identified by psyche;
```

Ora Dora ha la password “psyche” invece di “avocado.”

Gestione delle password

A partire da ORACLE8 le password possono scadere e gli account possono essere bloccati dopo ripetuti tentativi di connessione falliti. Quando si cambia la password, viene conservata una sua storia in modo da 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. Per dettagli completi su questo comando si può consultare la guida alfabetica di riferimento (Capitolo 37) al paragrafo dedicato a `CREATE PROFILE`. Per quel che riguarda le password e l'accesso di account, i profili possono imporre quanto segue.

- Una “durata” della password, che stabilisce con quale frequenza deve essere cambiata.

- 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 minima.

Per cambiare la password, oltre al comando `alter user`, si può utilizzare il comando `password` in SQLPLUS. Se si utilizza questo comando, la nuova password non viene visualizzata sullo schermo mentre viene digitata. Utenti con autorità di dba possono modificare la password di qualsiasi altro utente attraverso il comando `password`; gli altri utenti possono cambiare solo la propria password.

Quando si digita il comando `password`, vengono richieste la password vecchia e quella nuova, come mostrato nel seguente listato:

```
password
Changing password for dora
Old password:
New password:
Retype new password:
```

Quando il cambiamento di password riesce, si ottiene il messaggio di conferma:

```
Password changed
```

Tre ruoli standard

Ora che ha un account, che cosa può fare Dora in ORACLE? A questo punto nulla, infatti Dora non ha alcun privilegio di sistema.

ORACLE fornisce tre ruoli standard per compatibilità con le versioni precedenti: CONNECT, RESOURCE e DBA.

Il ruolo CONNECT Agli utenti occasionali, soprattutto a quelli che non hanno necessità di creare tabelle, di solito viene assegnato solo il ruolo CONNECT. Questo consiste semplicemente nel privilegio di utilizzare ORACLE. Questo diritto diventa significativo con l'aggiunta dell'accesso a particolari tabella che appartengono ad altri utenti e con la concessione dei privilegi di selezionare, inserire, aggiornare e cancellare righe in queste tabelle. Gli utenti che hanno il ruolo CONNECT possono anche creare tabelle, viste, sequenze, cluster, sinonimi (trattati più avanti), sessioni (spiegate nella guida alfabetica di riferimento del Capitolo 37) e link ad altri database (trattati nel Capitolo 21).

Il ruolo RESOURCE A utenti del database più esigenti e regolari può essere concesso il ruolo RESOURCE. Questo concede anche i diritti per creare proprie tabelle, sequenze, procedure, trigger, indici e cluster (gli ultimi due sono trattati nel Capitolo 19; per una trattazione dei trigger si deve consultare il Capitolo 23 e per le procedure il Capitolo 24).

Il ruolo DBA Il ruolo DBA (amministratore del database) ha tutti i privilegi di sistema, comprese quote di spazio illimitate, e la capacità di concedere tutti i privilegi ad altri utenti. In questo capitolo, dba si riferisce alla persona che è 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 al dba non sono mai assegnati ai normali utenti, né sono loro necessari, perciò qui verranno trattati più brevemente. Altri diritti tipicamente utilizzati dai dba sono impiegati regolarmente anche dagli utenti e sono importanti. Questo sottosinsieme di privilegi dba sarà presentato tra poco. In ORACLE, al DBA sono concessi i ruoli EXP_FULL_DATABASE e IMP_FULL_DATABASE, che a loro volta possiedono i privilegi necessari per l'esportazione e l'importazione dell'intero database ORACLE.

Sintassi per il comando grant

La sintassi del comando grant per privilegi di sistema è la seguente:

```
grant {privilegio sistema | ruolo}
      [, {privilegio sistema | ruolo}, . . .]
      to {utente | ruolo} [, {utente | ruolo}], . . .
      [with admin option]
```

Utilizzando il comando grant si può concedere qualsiasi privilegio o ruolo a un altro utente o a un altro ruolo. La clausola with admin option consente al beneficiario di assegnare il privilegio o ruolo ad altri utenti o ruoli. Chi ha concesso il ruolo 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}
        [, {privilegio sistema | ruolo}, . . .]
        from {utente | ruolo} [, {utente | ruolo}], . . .
```

Un individuo con ruolo DBA può revocare CONNECT, RESOURCE, DBA o qualsiasi altro privilegio o ruolo a chiunque altro, compreso un altro dba. Questo 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 alla tabella possono accedervi esattamente come prima.

Per rimuovere un utente e tutte le risorse da lui possedute, occorre utilizzare il comando drop user:

```
drop user utente [cascade];
```

L'opzione cascade rimuove l'utente con tutti gli oggetti da lui posseduti, compresi i vincoli di integrità referenziale. Questa opzione annulla viste, sinonimi, procedure memorizzate, funzioni o package che si riferiscono a oggetti nello schema dell'utente rimosso. Se non viene impiegata e vi sono ancora oggetti posseduti dall'utente, ORACLE non rimuove l'utente e restituisce invece un messaggio di errore.

18.2 Che cosa possono concedere gli utenti

Un utente può concedere privilegi su qualsiasi oggetto da lui posseduto. Il dba può concedere qualsiasi privilegio di sistema (poiché il ruolo DBA ha i privilegi grant any privilege e grant any role).

Si supponga che l'utente Dora, che possiede la tabella COMFORT ed è un dba, crei due nuovi utenti, Bob e Judy, con i seguenti privilegi:

```
create user Judy identified by sarah;
```

User created.

```
grant CONNECT to Judy;
```

Role granted.

```
create user Bob identified by carolyn;
```

User created.

```
grant CONNECT, RESOURCE to bob;
```

Role granted.

Questa sequenza di comandi assegna sia a Judy sia a Bob la possibilità di connettersi a ORACLE, e concede a Bob alcune ulteriori capacità. Ma questi utenti non possono fare nulla con le tabelle di Dora, senza un accesso esplicito.

Per concedere ad altri utenti l'accesso alle proprie tabelle, si deve utilizzare una seconda forma del comando grant:

```
grant privilegio oggetto [(colonna [, colonna])]  
on oggetto to {utente | ruolo}  
[with grant option];
```

I privilegi che un utente può concedere comprendono i seguenti.

- Su tabelle e viste (soltanto quelle possedute dall'utente):
 - INSERT
 - UPDATE (tutte le colonne o solo alcune particolari)
 - DELETE
- Soltanto sulle tabelle:
 - ALTER (tabelle, tutte le colonne o solo alcune particolari, o sequenze)
 - REFERENCES
 - INDEX (colonne in una tabella)
 - ALL (tutti i privilegi elencati sopra)
- Su procedure, funzioni, package, tipi di dati astratti e librerie:
 - EXECUTE
- Su tabelle, viste, sequenze e snapshot:
 - SELECT
- Su directory (per tipi di dati BFILE LOB):
 - READ

Il privilegio oggetto concesso deve essere uno dei privilegi oggetto (ALL, ALTER, DELETE, EXECUTE, INDEX, INSERT, REFERENCES, SELECT o UPDATE). Questi privilegi offrono al beneficiario della concessione la capacità di intraprendere alcune azioni sugli oggetti. L'oggetto può essere una tabella, una vista, una sequenza, una procedura, una funzione, un package, uno snapshot o un sinonimo per uno qualsiasi di questi oggetti.

Quando si esegue una procedura o una funzione di un altro utente, questa viene eseguita utilizzando i privilegi del 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 sottostanti.

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 sulla sua tabella all'utente Bob, Bob 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 viste basate sulle tabelle di un altro utente e di concedere accessi a queste viste ad altri utenti ancora, è necessario avere un accesso with grant option sulle 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`. Come con altri prodotti ORACLE, si può fare questo con uno dei seguenti metodi: inserire il nome utente e la password sulla stessa riga del comando, digitare il comando da solo e poi rispondere alle richieste oppure digitare il comando e il nome utente e poi rispondere alla richiesta della password.

Gli ultimi due metodi evitano la visualizzazione della password e sono perciò più sicuri.

```
connect Bob/carolyn
```

Connected.

Una volta connessa, Dora può effettuare selezioni dalla tabella su cui ha concesso a Bob l'accesso `SELECT`.

NOTA *Il nome della tabella deve essere preceduto 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-93 | 16.9 | 5.7 | .5 | |
| SAN FRANCISCO | 22-JUN-93 | 10.6 | 22.2 | .1 | |
| SAN FRANCISCO | 23-SEP-93 | | 16.4 | .1 | |
| SAN FRANCISCO | 22-DEC-93 | 11.4 | 4.3 | 2.3 | |
| KEENE | 21-MAR-93 | 4.4 | -18 | 4.4 | |
| KEENE | 22-JUN-93 | 29.5 | 19.3 | 1.3 | |
| KEENE | 23-SEP-93 | 37.7 | 28.1 | | |
| KEENE | 22-DEC-93 | -22 | -18 | 3.9 | |

Per convenienza 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-93 | 16.9 | 5.7 | .5 | |
| SAN FRANCISCO | 22-JUN-93 | 10.6 | 22.2 | .1 | |
| SAN FRANCISCO | 23-SEP-93 | | 16.4 | .1 | |
| SAN FRANCISCO | 22-DEC-93 | 11.4 | 4.3 | 2.3 | |
| KEENE | 21-MAR-93 | 4.4 | -18 | 4.4 | |

| | | | | |
|-------|-----------|------|------|-----|
| KEENE | 22-JUN-93 | 29.5 | 19.3 | 1.3 |
| KEENE | 23-SEP-93 | 37.7 | 28.1 | |
| KEENE | 22-DEC-93 | -22 | -18 | 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.

Poi 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-0942: table or view does not exist

Il tentativo di effettuare selezioni dalla vista COMFORT non riesce perché la tabella sottostante di questa vista di Bob è Dora.COMFORT. Ovviamente un tentativo di selezione dalla tabella Dora.COMFORT produce lo stesso messaggio. Poi viene effettuato un tentativo di selezione da Dora.QUALCHECOMFORT:

```
select * from Dora.QUALCHECOMFORT;
```

| CITTA | DATA | CAMPIONE | MEZZOGIORNO | MEZZANOTTE | PRECIPITAZIONE |
|-------|-----------|----------|-------------|------------|----------------|
| KEENE | 21-MAR-93 | 4.4 | -18 | 4.4 | |
| KEENE | 22-JUN-93 | 29.5 | 19.3 | 1.3 | |
| KEENE | 23-SEP-93 | 37.7 | 28.1 | | |
| KEENE | 22-DEC-93 | -22 | -18 | 3.9 | |

Questo funziona perfettamente, anche se è stato revocato l'accesso diretto alla tabella COMFORT, 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 sottostanti. Gli utenti così sono in grado di vedere solo le informazioni presentate dalla vista. Il meccanismo può essere esteso in modo che sia specifico per ogni utente. Al termine di questo capitolo viene fornita una spiegazione dettagliata di questa caratteristica.

Ora viene creata la vista POCOCOMFORT sulla vista QUALCHECOMFORT, sotto il nome utente Bob:

```
create view POCOCOMFORT as select * from Dora.QUALCHECOMFORT;
```

View created.

e la riga per il 23 settembre 1993 viene aggiornata:

```
update POCOCOMFORT set Mezzogiorno = 31.1
where DataCampione = TO_DATE('23-SEP-1993','DD-MON-YYYY');
```

1 row updated.

Quando viene effettuata una query sulla vista POCOCOMFORT, sono visibili gli effetti dell'aggiornamento:

```
select * from POCOCOMFORT;
```

| CITTA | DATACAMPIONE | MEZZOGIORNO | MEZZANOTTE | PRECIPITAZIONE |
|-------|--------------|-------------|------------|----------------|
| KEENE | 21-MAR-93 | 4.4 | -18 | 4.4 |
| KEENE | 22-JUN-93 | 29.5 | 19.3 | 1.3 |
| KEENE | 23-SEP-93 | 31.1 | 28.1 | |
| KEENE | 22-DEC-93 | -22 | -18 | 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 sottostante, 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. Questo si adatta allo standard ANSI che si sta sviluppando e riflette il fatto che un utente che ha solo privilegi *UPDATE* o *DELETE* su una tabella potrebbe utilizzare i commenti di ritorno del database per scoprire informazioni sui dati sottostanti.

create synonym

Un metodo alternativo alla creazione di una vista che include un'intera tabella o vista di un altro utente è quello di creare sinonimi:

```
create synonym POCOCOMFORT for Dora.QUALCHECOMFORT;
```

Questo sinonimo può essere creato esattamente come una vista. Per informazioni si può consultare il paragrafo dedicato al comando `create synonym` nella guida alfabetica di riferimento (Capitolo 37).

Utilizzo di privilegi non concessi

Si supponga che venga effettuato un tentativo di cancellare la riga che è stata appena aggiornata:

```
delete from POCOCOMFORT where DataCampione = '23-SEP-93';
*
ERROR at line 1: ORA-1031: insufficient privileges
```

A Bob non sono stati concessi da Dora i privilegi `DELETE`, perciò 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
```

Poiché Bob non ha questa autorità, non riesce a trasmetterla a Judy. Poi 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
```

Egli non può concedere questo privilegio perché la vista denominata `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 può cercare di trasmettere qualche 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'
```

Dal momento che `POCOCOMFORT` si basa su una delle viste di Dora e a Bob non è stato concesso `SELECT` with grant option, la concessione di Bob non riesce.

In seguito viene creata una nuova tabella, posseduta da Bob, caricata con le informazioni correnti della vista `POCOCOMFORT`:

```
create table NOCOMFORT as
select * from POCOCOMFORT;
```

Table created.

Poi vengono concessi a Judy i privilegi SELECT su di essa:

```
grant select on NOCOMFORT to Judy;
```

Grant succeeded.

Per controllare questa concessione, ci si connette al nome utente di Judy, in questo modo:

```
connect Judy/sarah
Connected.
```

Quindi vengono effettuate delle query sulla tabella per cui Bob ha concesso a Judy il privilegio SELECT:

```
select * from Bob.NOCOMFORT;
```

| CITTA | DATA | CAMPIONE | MEZZOGIORNO | MEZZANOTTE | PRECIPITAZIONE |
|-------|-----------|----------|-------------|------------|----------------|
| KEENE | 21-MAR-93 | | 4.4 | -18 | 4.4 |
| KEENE | 22-JUN-93 | | 29.5 | 19.3 | 1.3 |
| KEENE | 23-SEP-93 | | 37.7 | 28.1 | |
| KEENE | 22-DEC-93 | | -22 | -18 | 3.9 |

Questa concessione è riuscita. È stata creata una tabella NOCOMFORT dal nome utente di Bob, che quindi la possiede. Egli è stato perciò 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:

```
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 noti 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-0942: table or view does not exist
```

Questo messaggio non comunica che non si ha il privilegio di accesso, per fare in modo che una persona che non ha il permesso di effettuare una query su una tabella non sappia neppure se questa esiste.

Creazione di un ruolo

Oltre ai tre ruoli di sistema mostrati in precedenza in questo capitolo, CONNECT, RESOURCE e DBA, è possibile creare propri ruoli. I ruoli creati possono comprendere privilegi di tabelle o di sistema o una combinazione di entrambi. Nei paragrafi seguenti viene mostrato come creare e amministrare 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 seguente listato:

```
create role nome_ruolo
[not identified|identified [by password|externally]];
```

Un ruolo appena creato non ha associato alcun privilegio. Le opzioni di password per la creazione di ruoli sono trattate nel paragrafo “Aggiunta di una password a un ruolo” in questo capitolo.

Nel seguente esempio vengono mostrati due comandi create role:

```
create role IMPIEGATO;
create role MANAGER;
```

I due comandi creano i ruoli IMPIEGATO e MANAGER, che verranno utilizzati negli esempi dei paragrafi seguenti.

Concessione di privilegi a un ruolo

Una volta che il ruolo è stato creato, è possibile concedergli privilegi. La sintassi del comando grant per i ruoli è la stessa di 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 seguente listato:

```
grant select on COMFORT to IMPIEGATO;
```

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 privilegi di sistema, come CREATE SESSION, CREATE SYNONYM e CREATE VIEW, ai ruoli. Questi privilegi saranno quindi utilizzabili da ogni utente del ruolo.

La capacità di collegarsi al database è concessa attraverso il privilegio CREATE SESSION. Nell'esempio seguente viene concesso questo privilegio al ruolo IMPIEGATO. Questo privilegio viene concesso anche al ruolo MANAGER, insieme con il 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, attraverso il comando grant, come mostrato nel codice 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 a un utente, come viene spiegato nel paragrafo seguente) si può utilizzare la clausola with admin option, come mostrato nel seguente listato:

```
grant IMPIEGATO to MANAGER with admin option;
```

Con questa clausola, il destinatario della concessione ha l'autorità di concedere il ruolo ad altri utenti o ruoli e può anche modificare o rimuovere il ruolo stesso.

Concessione di un ruolo a utenti È possibile concedere ruoli a utenti. In questo caso i ruoli possono considerati come insiemi di privilegi a cui è stato assegnato un nome. Invece di concedere ciascun privilegio a ciascun utente, si concedono i privilegi al ruolo e il ruolo a ogni utente. Questo semplifica enormemente i compiti amministrativi compresi nella gestione dei privilegi.

NOTA *I privilegi concessi a 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;
```

Qui l'utente Bob ha tutti i privilegi concessi al ruolo IMPIEGATO (cioè 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 seguente listato:

```
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: cambiare l'autorità necessaria a concederlo. Per default, i ruoli non hanno associata una 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 si può utilizzare la clausola identified by nel comando alter role per specificare una password, come mostrato nel seguente listato:

```
alter role MANAGER identified by cygnusxi;
```

Ogni volta che un utente cerca di attivare questo ruolo, viene richiesta la password. Se però quel ruolo è impostato come ruolo di default per l'utente, non è richiesta alcuna password quando l'utente si connette. Per ulteriori dettagli su questo argomento si può consultare il paragrafo “Attivazione e disattivazione di ruoli” in questo capitolo.

I ruoli possono essere legati ai privilegi del sistema operativo. Se questa capacità è disponibile, allora si può utilizzare nel comando alter role la clausola identified externally. 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;
```

In VMS il processo di verifica utilizza gli identificatori di diritti del sistema operativo. Nella maggior parte dei sistemi UNIX il processo di verifica impiega il file denominato /etc/group. Per utilizzare questa caratteristica in altri sistemi operativi, si deve porre a TRUE il parametro OS_ROLES di avvio del database nel file init.ora.

Il seguente esempio mostra il processo di verifica per un'istanza del database chiamata "Local" su un sistema UNIX. Il file /etc/group del server può contenere la seguente annotazione:

```
ora_local_manager_d:NONE:1:dora
```

Questa annotazione 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 stato anche quello di default 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 all'annotazione di /etc/group, come mostrato nel seguente listato:

```
ora_local_manager_d:NONE:1:dora,judy
```

Se si utilizza questa opzione, 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 seguente listato. Per default, i ruoli non hanno password.

```
alter role MANAGER not identified;
```

Attivazione e disattivazione di ruoli

Quando l'account di un utente viene modificato, può essere creato un elenco di ruoli di default, attraverso la clausola default role del comando alter user. L'azione di default di questo comando imposta tutti i ruoli di un utente come ruoli di default, attivandoli tutti ogni volta che l'utente si connette.

La sintassi per questa parte del comando alter user è la seguente:

```
alter user nomeutente  
default role {[ruolo1, ruolo2]  
[all|all except ruolo1, ruolo2] [NONE]};
```

Come mostra questa sintassi, un utente può essere modificato in modo da avere attivati per default particolari ruoli, tutti i ruoli, tutti i ruoli tranne alcuni particolari o nessun ruolo.

```
alter user Bob
default role IMPIEGATO;
```

Per attivare un ruolo non di default, 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 ha associata una password, questa deve 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 seguente listato, disattivando tutti i ruoli nella sessione corrente. Quindi occorre attivare di nuovo quelli che si desiderano.

```
set role none;
```

Dal momento che potrebbe essere necessario eseguire di quando in quando 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. Si deve specificare il privilegio, il nome dell'oggetto (se è un privilegio di oggetto) e il nome del ruolo, come mostrato nel seguente esempio:

```
revoke SELECT on COMFORT from IMPIEGATO;
```

Ora gli utenti del ruolo IMPIEGATO non possono più effettuare query sulla tabella COMFORT.

Scaricamento di un ruolo

Per scaricare un ruolo, occorre utilizzare il comando drop role, come mostrato nel seguente esempio:

```
drop role MANAGER;
drop role IMPIEGATO;
```

I ruoli specificati, e i loro privilegi associati, verranno rimossi completamente dal database.

Concessione di update per particolari colonne

In alcuni casi si desidera concedere agli utenti il privilegio SELECT su più colonne di quelle su cui si vuole concedere il privilegio UPDATE. Dal momento che le colonne di SELECT possono essere ristrette attraverso una vista, per restringere ulteriormente le colonne che possono essere aggiornate è necessaria una particolare forma del comando grant. Ecco un esempio per due colonne di COMFORT:

```
grant update (Mezzogiorno, Mezzanotte) on COMFORT to Judy;
```

Revoca di privilegi

Se i privilegi di oggetto possono essere concessi, possono anche essere annullati, con una sintassi simile a quella del comando grant:

```
revoke privilegio oggetto [, privilegio oggetto . . . ]  
on oggetto  
from {utente | ruolo} [, {utente | ruolo}]  
[cascade constraints];
```

revoke all rimuove tutti i privilegi elencati precedentemente, da SELECT a INDEX; la revoca di singoli privilegi lascia inalterati gli altri che sono stati concessi. L'opzione with grant option viene revocata insieme con il privilegio a cui è associata.

Se un utente definisce vincoli di integrità referenziale sull'oggetto, per rimuoverli occorre revocare i privilegi sull'oggetto utilizzando l'opzione cascade constraints.

Sicurezza con User

L'accesso alle tabelle può essere concesso in modo specifico, tabella per tabella e vista per vista, a ogni utente. Vi è comunque un'altra tecnica che in alcuni casi semplifica questo processo. Occorre qui riprendere la tabella LAVORATORE di Talbot:

```
select * from LAVORATORE;
```

| NOME | ETA ALLOGGIO |
|------------------|---------------|
| ADAH TALBOT | 23 PAPA KING |
| ANDREW DYE | 29 ROSE HILL |
| BART SARJEANT | 22 CRANMER |
| DICK JONES | 18 ROSE HILL |
| DONALD ROLLO | 16 MATTS |
| ELBERT TALBOT | 43 WEITBROCHT |
| GEORGE OSCAR | 41 ROSE HILL |
| GERHARDT KENTGEN | 55 PAPA KING |

| | |
|---------------------------|--------------|
| HELEN BRANDT | 15 |
| JED HOPKINS | 33 MATT |
| JOHN PEARSON | 27 ROSE HILL |
| KAY AND PALMER WALLBOM | ROSE HILL |
| PAT LAVAY | 21 ROSE HILL |
| PETER LAWSON | 25 CRANMER |
| RICHARD KOCH AND BROTHERS | WEITBROCHT |
| ROLAND BRANDT | 35 MATT |
| VICTORIA LYNN | 32 MULLERS |
| WILFRED LOWELL | 67 |
| WILLIAM SWING | 15 CRANMER |

Per consentire a ogni lavoratore 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 lavoratore. In alternativa si può creare una vista la cui clausola where contiene User, la pseudocolonna, in questo modo:

```
create view TUAETA as
select * from LAVORATORE
where SUBSTR(Nome,1,INSTR(Nome,' ')-1) = User;
```

View created.

Quando un utente di nome George crea questa vista e effettua una query nella tabella LAVORATORE attraverso la vista TUAETA, la clausola where trova il suo nome utente, GEORGE, nella colonna Nome (si veda la clausola where) e produce questo:

```
select * from TUAETA;
```

| NOME | ETA ALLOGGIO |
|--------------|--------------|
| GEORGE OSCAR | 41 ROSE HILL |

Ora George concede la possibilità di effettuare selezioni in questa vista a Bart Sarjeant:

```
grant SELECT on TUAETA to Bart;
```

Poi si connette a Bart per controllare gli effetti:

```
connect Bart/stjohn
Connected.
```

```
select * from George.TUAETA;
```

| NOME | ETA ALLOGGIO |
|---------------|--------------|
| BART SARJEANT | 22 CRANMER |

Curiosamente Bart ottiene la propria riga, non quella di George, perché la pseudocolonna User, nella vista di George, è sempre uguale all'utente di SQLPLUS al momento in cui viene effettuata la query nella vista.

Concessione dell'accesso al pubblico

Invece di concedere l'accesso a ogni lavoratore, il comando grant può essere generalizzato a public:

```
grant select on TUAETA to public;
```

Questo dà l'accesso a tutti, compresi gli utenti creati dopo che questa concessione è stata effettuata. Tuttavia, ognuno deve ancora accedere alla tabella utilizzando come prefisso il nome utente di George. Per evitare questo, un dba potrebbe creare un sinonimo pubblico (un nome accessibile a tutti gli utenti che sostituisce George.TUAETA).

```
create public synonym TUAETA for George.TUAETA;
```

Da questo momento, chiunque può accedere a TUAETA senza utilizzare il prefisso George. Questo approccio dà enorme flessibilità per la sicurezza. I lavoratori potrebbero vedere solo il proprio salario, ad esempio, in una tabella che contiene i salari di tutti. Se però 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.

18.3 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 seguente elenco. In questo esempio, a Bob viene concessa una quota di 100 MB nel tablespace USERS.

```
alter user Bob
quota 100M on USERS;
```

Una quota di spazio di utente può essere stabilita quando l'utente viene creato, attraverso il comando create user. Se non vi sono limiti nella quota di spazio dell'utente, si può concedere all'utente il privilegio di sistema UNLIMITED TABLESPACE. Per ulteriori dettagli su questi comandi si possono consultare create user e alter user nella guida alfabetica di riferimento del Capitolo 37.

• Capitolo 19

• **Modifica dell'ambiente di ORACLE**

• 19.1 **Indici**

• 19.2 **Tablespace e struttura del database**

• 19.3 **Cluster**

• 19.4 **Sequenze**

Come nel Capitolo 18, in questo capitolo viene esaminato un sottoinsieme di funzioni per l'amministrazione del database (DBA) utilizzato dalla maggior parte degli utenti, non ristretto ai dba. Queste funzioni includono indici sulle tabelle, cluster, generatori di sequenze e allocazioni di spazio per tabelle e indici. In questo capitolo viene mostrato come sono strutturati internamente i database di ORACLE, aiutando a capire come funzionano realmente alcune caratteristiche e come sono intercorrelate. Vengono presentate solo alcune opzioni dei comandi `create index`, `create tablespace` e `create cluster`, quelle complete sono mostrate nella guida alfabetica di riferimento del Capitolo 37.

19.1 **Indici**

Un *indice* è un concetto semplice. È tipicamente un elenco di parole chiave accompagnate dalla *posizione* dell'informazione su un particolare argomento. Per trovare informazioni sugli indici, ad esempio, occorre cercare la parola “indici” nell'indice analitico di questo libro; così si trova il numero di questa pagina. La parola “indici” è la chiave e i numeri di pagina indicano dove localizzare la trattazione sugli 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 lavoro. Ad esempio, anche se è possibile trovare informazioni sugli indici semplicemente sfogliando il libro finché non si trova questa pagina, questo processo è lento e impiega molto tempo. Poiché l'indice analitico alla fine del libro è in ordine alfabetico, si può andare velocemente al punto appropriato nell'indice (senza leggere tutte le voci) dove si trova la voce “indici”. Questo è ovviamente più rapido che leggere il libro dall'inizio. Questi stessi principi si applicano agli indici di ORACLE.

Si consideri la tabella HOCKEY:

```
select * from HOCKEY;
```

| SQUADRA | VINTA | PERSA | PARI |
|--------------|-------|-------|------|
| Quebec | 6 | 23 | 4 |
| Detroit | 10 | 18 | 5 |
| Vancouver | 11 | 16 | 6 |
| NY Islanders | 11 | 20 | 4 |
| Washington | 13 | 15 | 4 |
| Pittsburgh | 13 | 16 | 3 |
| Calgary | 14 | 11 | 9 |
| St. Louis | 14 | 12 | 6 |
| Winnipeg | 14 | 13 | 5 |
| NY Rangers | 15 | 14 | 5 |
| New Jersey | 15 | 15 | 3 |
| Edmonton | 16 | 11 | 7 |
| Philadelphia | 16 | 14 | 4 |
| Los Angeles | 16 | 14 | 3 |
| Hartford | 16 | 17 | 1 |
| Toronto | 16 | 18 | 0 |
| Boston | 17 | 13 | 3 |
| Minnesota | 17 | 15 | 2 |
| Chicago | 19 | 13 | 2 |
| Montreal | 20 | 13 | 4 |
| Buffalo | 21 | 9 | 4 |

Poche tabelle che si utilizzano nella pratica reale sono tanto brevi e raramente sono in ordine alfabetico. La seguente query chiede a ORACLE di cercare una particolare squadra in base al numero di vittorie:

```
select Squadra, Vinta, Persa, Pari
      from HOCKEY
     where Vinta = 20;
```

Se la tabella HOCKEY non ha un indice sulla colonna Vinta, ORACLE deve leggere ogni riga nella tabella finché non trova tutte le squadre che corrispondono alla clausola `where` della query.

Per velocizzare questo processo, si può creare un indice sulla colonna Vinta. Poi, quando si esegue la stessa query, ORACLE cerca prima nell'indice, che è in ordine numerico, trovando così le squadre con 20 vittorie molto più velocemente (ORACLE non legge ogni voce, ma salta direttamente nelle vicinanze della squadra, proprio come si fa quando si consulta l'indice di un libro).

La voce dell'indice fornisce a ORACLE la locazione esatta nella tabella (e sul disco) della riga Montreal.

Sapendo questo, è chiaro che inserire in un indice una colonna importante (una che probabilmente sarà presente in una clausola `where`) velocizza la risposta di ORACLE a una query. Velocizza anche query dove vengono unite due tabelle, se le colonne a 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.

Creazione di un indice

Questo comando è il metodo fondamentale per creare indici:

```
create [bitmap] [unique] index indice on tabella(colonna [,colonna] . . .);
```

indice deve essere un nome unico e seguire la convenzione di assegnazione dei nomi 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, disponibili a partire da ORACLE7.3, consentono di creare utili indici su colonne con pochissimi valori distinti; per questo occorre leggere il paragrafo “Creazione di indici bitmap” più avanti in questo capitolo.

Si può costruire un indice singolo su più colonne elencando le colonne una di seguito all'altra, separate da virgole. A questo punto è opportuno riprendere la tabella COMPITOLAVORATORE. La chiave primaria di questa tabella è una combinazione di Nome e Compito. La seguente query produce la tabella mostrata nella parte superiore della Figura 19.1:

```
select RowID, Nome, Compito, Capacita
      from COMPITOLAVORATORE;
```

Per questo esempio è stata selezionata anche RowID, che corrisponde alla locazione interna di una riga (come un numero di pagina in un libro) utilizzata da ORACLE quando salva le righe di dati in una tabella. Ora viene creato un indice sulla chiave primaria

```
create index COMPITOLAVORATORE_NOME_COMPITO on LAVORATORE(Nome,Compito);
```

Questa è una tecnica utile e pratica per assegnare un nome all'indice, combinando i nomi della tabella e delle colonne, fino a 30 caratteri. Non è necessario conoscere il nome dell'indice per un query o altri comandi SQL, perché ORACLE lo utilizza ogni volta che è possibile, ma quando si elencano (o rimuovono) gli indici creati (i loro nomi sono salvati nella vista del dizionario di dati USER_INDEXES, su cui è possibile effettuare query), dai loro nomi si capisce immediatamente a che cosa servono. L'indice in ORACLE appare come la seconda tabella illustrata nella Figura 19.1.

Quando si esegue una query come questa:

```
select Nome, Compito, Capacita
      from COMPITOLAVORATORE
     where Nome = 'JOHN PEARSON'
       and Compito = 'TAGLIALEGNA';
```

ORACLE cerca Nome e Compito nell'indice e prende RowID (AAAAs-YABQAAAAGzAAE) e poi le informazioni corrispondenti a RowID nella tabella COMPITOLAVORATORE.

Tabella COMPITOLAVORATORE con RowID per ogni riga:

| ROWID | NOME | COMPITO | CAPACITA |
|--------------------|----------------|-------------|--------------|
| AAAAsYABQAAAAGzAAA | DICK JONES | FABBRO | ECCELLENTE |
| AAAAsYABQAAAAGzAAB | JOHN PEARSON | TREBBIATORE | |
| AAAAsYABQAAAAGzAAC | JOHN PEARSON | FABBRO | MEDIO |
| AAAAsYABQAAAAGzAAD | HELEN BRANDT | TREBBIATORE | MOLTO VELOCE |
| AAAAsYABQAAAAGzAAE | JOHN PEARSON | TAGLIALEGNA | BUONO |
| AAAAsYABQAAAAGzAAF | VICTORIA LYNN | FABBRO | PRECISO |
| AAAAsYABQAAAAGzAAG | ADAH TALBOT | OPERAIO | BUONO |
| AAAAsYABQAAAAGzAAH | WILFRED LOWELL | OPERAIO | MEDIO |
| AAAAsYABQAAAAGzAAI | ELBERT TALBOT | ARATORE | LENTO |
| AAAAsYABQAAAAGzAAJ | WILFRED LOWELL | ARATORE | MEDIO |

Indice sulla tabella COMPITOLAVORATORE con RowID della tabella:

| | | |
|----------------|-------------|--------------------|
| ADAH TALBOT | OPERAIO | AAAAsYABQAAAAGzAAG |
| DICK JONES | FABBRO | AAAAsYABQAAAAGzAAA |
| ELBERT TALBOT | ARATORE | AAAAsYABQAAAAGzAAI |
| HELEN BRANDT | TREBBIATORE | AAAAsYABQAAAAGzAAD |
| JOHN PEARSON | TREBBIATORE | AAAAsYABQAAAAGzAAB |
| JOHN PEARSON | FABBRO | AAAAsYABQAAAAGzAAC |
| JOHN PEARSON | TAGLIALEGNA | AAAAsYABQAAAAGzAAE |
| VICTORIA LYNN | FABBRO | AAAAsYABQAAAAGzAAF |
| WILFRED LOWELL | ARATORE | AAAAsYABQAAAAGzAAJ |
| WILFRED LOWELL | OPERAIO | AAAAsYABQAAAAGzAAH |

Figura 19.1 La tabella COMPITOLAVORATORE e il suo indice.

L'indice è in un formato descritto come *B-tree*, che significa che è organizzato in una struttura ad albero, con nodi e foglie (come le mucche nel Capitolo 12) e si ri-struttura automaticamente ogni volta che viene inserita una nuova riga nella tabella. Una trattazione del meccanismo B-tree va al di là degli scopi di questo libro, ma può essere trovata in molti libri di informatica.

Il prossimo paragrafo si occupa di un certo numero di problemi relativi all'utilizzo degli indici.

Imposizione dell'unicità

Nel Capitolo 2 si è spiegato che una serie di tabelle è detta in terza forma normale se tutte le colonne in una riga sono dipendenti soltanto dalla chiave primaria. Nella tabella COMPITOLAVORATORE la chiave primaria è la combinazione di Nome e Compito. 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 critica. In una banca con numeri di conto duplicati o in un sistema di fatturazione con identificativi di clienti duplicati si avrebbero gravi problemi, dal momento che le transazioni potrebbero essere 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 chiave primaria, vi sono troppi doppioni). Per evitare questo pericolo, il database aiuta a impedire la creazione di chiavi primarie duplicate. ORACLE offre due agevolazioni: si può garantire l'unicità di una chiave attraverso l'inserimento in indici o attraverso vincoli, oppure si possono utilizzare i generatori di sequenze, trattati più avanti in questo capitolo.

Creazione di un indice unico Per creare un indice che garantisca l'unicità della chiave primaria (sia una singola colonna che più colonne), come sulla tabella COMPITOLAVORATORE, occorre utilizzare il vincolo di chiave primaria sulle colonne della chiave nell'istruzione `create table`. Si può anche impiegare l'istruzione `create unique index`, ma questa fallisce se esiste già qualche duplicato. Se l'istruzione `create unique index` riesce, qualsiasi tentativo 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-0001: duplicate value in index
```

Possono esistere casi in cui si vuole imporre l'unicità non in una chiave primaria; questo è possibile attraverso il vincolo `unique`. Ad esempio, se si include un codice fiscale per ogni persona, ma la chiave primaria è una sequenza, si può garantire l'unicità della colonna Codice fiscale con un vincolo `unique`.

Si consideri ad esempio la tabella AZIONE. La chiave primaria è la colonna Società. Però anche la colonna Simbolo dovrebbe essere unica, benché non sia una chiave primaria.

Per imporre l'unicità su entrambe queste colonne, occorre creare due vincoli distinti quando si crea la tabella, come mostrato nel seguente listato.

```
create table AZIONE (
  Societa      VARCHAR2(20) primary key,
  Simbolo      VARCHAR2(6) unique,
  Industria    VARCHAR2(15),
  ChiusIeri    NUMBER(6,2),
  ChiusOggi    NUMBER(6,2),
  Volume       NUMBER);
```

Quando si creano i vincoli di chiave primaria e unico, specificati per la tabella AZIONE, ORACLE crea automaticamente indici unici per imporre questi vincoli. 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, dal momento che accrescono il costo delle transazioni di manipolazione dei dati sulle tabelle di cui costituiscono l'indice.

Gli indici bitmap sono appropriati quando colonne non selettive sono impiegate come condizioni limitative in una query. Ad esempio, se vi sono pochissimi valori Alloggio distinti in una tabella LAVORATORE molto grande, di solito non è possibile creare indici B-tree su Alloggio, anche se tale colonna è frequentemente utilizzata nella clausola where. Tuttavia, Alloggio potrebbe trarre vantaggio da un indice bitmap.

Internamente, un indice bitmap mappa i valori distinti delle colonne su tutti i record. In questo esempio, si supponga che vi siano solo due valori di Alloggio ('WEITBROCHT' e 'MATTES') in una grande tabella LAVORATORE. Dal momento che vi sono solo due valori di Alloggio, vi sono due distinte voci per l'indice bitmap di Alloggio.

Se le prime cinque righe nella tabella hanno un valore di Alloggio di 'MATTES' e le cinque righe successive di 'WEITBROCHT', le voci bitmap di Alloggio sono come quelle mostrate nel seguente listato:

Lodging bitmaps:

```
MATTES:      < 1 1 1 1 1 0 0 0 0 >
WEITBROCHT: < 0 0 0 0 1 1 1 1 >
```

Ciascun numero rappresenta una riga nella tabella ALLOGGIO. Poiché sono state considerate dieci righe, sono mostrati dieci valori bitmap. Leggendo la bitmap per Alloggio, i primi cinque record hanno un valore 'MATTES' (i valori '1') e i cinque successivi no (i valori '0'). In una colonna si potrebbero avere più di due valori, nel qual caso vi sarebbero voci bitmap differenti per ognuno.

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 seguente listato. Conviene indicare il fatto che si tratta di un indice di bitmap nel nome, in modo che possa essere rilevato facilmente durante le operazioni di regolazione.

```
create bitmap index LAVORATORE$BITMAP_ALLOGGIO
    on LAVORATORE(Alloggio);
```

Se si decide di utilizzare indici bitmap, è necessario considerare i vantaggi di 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 durante ogni transazione.

Non conviene utilizzare indici bitmap su una colonna a cui vengono aggiunti frequentemente nuovi valori. Ogni aggiunta di un nuovo valore alla colonna Alloggio richiede la creazione di una nuova bitmap.

Quando creare un indice

Gli indici sono utili soprattutto su tabelle molto grandi, su colonne che probabilmente saranno presenti in clausole where con una semplice uguaglianza, come la seguente:

```
where Nome = 'JOHN PEARSON'  
and Compito = 'TAGLIALEGNA'
```

o in unioni, come questa:

```
where LAVORATORE.Alloggio = ALLOGGIO.Alloggio
```

Questi indici consentono anche ricerche più veloci per le colonne incluse nell'indice presenti nelle clausole where, eccetto quelle che utilizzano IS NOT NULL e IS NULL. Se non vi è alcuna clausola where, non viene utilizzato alcun indice. Né distinct né group by attualmente utilizzano gli indici, ma order by può impiegarli (come viene spiegato più dettagliatamente più avanti in questo capitolo).

Quando gli indici vengono ignorati

ORACLE ha necessità di sapere che sta lavorando su una colonna semplice per la quale può rilevare un indice. Se una colonna inclusa in un indice viene modificata da una funzione, come SUBSTR o ||, ORACLE non impiega l'indice. Mentre con MIN e MAX, se utilizzati da sole su una colonna, è possibile servirsi dell'indice.

Varietà nelle colonne incluse in indici

Gli indici tradizionali (B-tree) sono utili soprattutto in colonne con una significativa varietà di dati. Ad esempio, una colonna che indica se un'azienda ditta attualmente è tra i clienti con S o N non va molto bene per un indice tradizionale e potrebbe in realtà rallentare una query. Una colonna di numeri di telefono andrebbe bene, una di prefissi telesellettivi sarebbe abbastanza marginale.

Quando una chiave primaria comprende più di una colonna, è meglio inserire nel vincolo di chiave primaria prima la colonna con la maggiore varietà. Se le colonne hanno una varietà relativamente simile, è opportuno mettere prima quella a cui probabilmente si accederà più spesso.

Conviene lasciare le piccole tabelle senza indici, eccetto che per imporre l'unicità della chiave primaria. Una piccola tabella in genere contiene meno di 30 righe, ma in una particolare applicazione può essere considerata piccola anche una tabella con 100 o più righe. A parte questo, la realizzazione di indici è quasi sempre vantaggiosa.

D'altra parte, gli indici bitmap offrono un'attuabile alternativa di indicizzazione per colonne che hanno pochissimi valori distinti. Gli indici bitmap vengono comunemente utilizzati per colonne di "flag" che sono limitate ai valori 'S' e 'N'.

Gli indici bitmap sono particolarmente efficaci quando in una query ne sono utilizzati diversi; l'ottimizzatore può velocemente valutare le bitmap e determinare quali righe soddisfano tutti i criteri per cui gli indici bitmap sono disponibili.

Quanti indici utilizzare su una tabella

Per una tabella si possono includere in un solo indice fino a 16 colonne (o un massimo da 1000 a 2000 byte circa per indice, a seconda del sistema operativo). Lo svantaggio portato dall'inserimento di troppe colonne si vede nella velocità di inserimento di nuove righe: ogni indice deve avere 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. Di solito i vantaggi dell'indicizzazione sono maggiori rispetto al costo in termini di lavoro e spazio occupato.

Con l'eccezione degli *indici cluster* (trattati più avanti in questo capitolo), le righe in cui le colonne sono NULL non compaiono in un indice. Se ad esempio si include in un indice la colonna Mezzogiorno della tabella COMFORT, come mostrato di seguito:

```
select * from COMFORT;
```

| CITTA | DATACAMPIONE | MEZZOGIORNO | MEZZANOTTE | PRECIPITAZIONE |
|---------------|--------------|-------------|------------|----------------|
| SAN FRANCISCO | 21-MAR-93 | 16.9 | 5.7 | .5 |
| SAN FRANCISCO | 22-JUN-93 | 10.6 | 22.2 | .1 |
| SAN FRANCISCO | 23-SEP-93 | | 16.4 | .1 |
| SAN FRANCISCO | 22-DEC-93 | 11.4 | 4.3 | 2.3 |
| KEENE | 21-MAR-93 | 4.4 | -18 | 4.4 |
| KEENE | 22-JUN-93 | 29.5 | 19.3 | 1.3 |
| KEENE | 23-SEP-93 | 37.7 | 28.1 | |
| KEENE | 22-DEC-93 | -22 | -18 | 3.9 |

si avrebbe una voce per ogni riga, con l'eccezione del record di San Francisco 23-SEP-93, perché il valore di Mezzogiorno in questo caso è NULL.

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.

Dal momento che gli indici sono tipicamente utilizzati dalle clausole where che contengono un'uguaglianza, la riga con una colonna NULL non viene restituita dall'istruzione select (si ricordi che non vi è nulla uguale a NULL). Questo potrebbe essere l'effetto desiderato.

Ad esempio, si potrebbe conservare una colonna di commissioni per venditori, lasciandola impostata a NULL, invece che 0, se un venditore non ha alcuna commissione. Una query come questa:

```
select Nome, Commissione
      from COMMISSIONE
     where Commissione > 0;
```

potrebbe essere eseguita più velocemente perché i venditori che non hanno alcuna commissione non compaiono neppure nell'indice sulla colonna Commissione.

Posizionamento di un indice nel database

È possibile specificare dove deve essere posto un indice per una tabella, assegnando un particolare tablespace. Come si è accennato brevemente nel Capitolo 18, un tablespace è una sezione del disco dove sono salvate le tabelle e gli indici. Un database può avere molti tablespace, ciascuno con un proprio nome. Un indice dovrebbe essere posto in un tablespace che si trova su un disco fisicamente separato dal tablespace dei dati. Questo riduce i potenziali conflitti di disco tra i file dei tablespace.

Per specificare il tablespace dove porre un indice occorre utilizzare la normale istruzione `create index` seguita dalla parola `tablespace` e dal nome del tablespace, come mostrato qui:

```
create index compitolavoratore_nom_compito on LAVORATORE(Nome,Compito)
    tablespace GBTALBOT;
```

`GBTALBOT` è il nome dato al tablespace creato precedentemente dall'amministratore del database. L'utilizzo dell'opzione `tablespace` in un'istruzione `create index` è di solito necessario solo per grandi tabelle dove l'ottimizzazione o l'utilizzo dello spazio sono critici. Se si ritiene che sia utile è opportuno consultarsi con l'amministratore del database.

Quando si crea un vincolo di chiave primaria o unico, ORACLE crea automaticamente un indice per imporre l'unicità. A meno che non si specifichi diversamente, l'indice viene creato nello stesso tablespace della tabella e utilizza i parametri di default per la memorizzazione di quel tablespace. Poiché la locazione dell'area di memorizzazione è spesso inadeguata, conviene utilizzare la clausola `using index` quando si creano vincoli di chiave primaria e unici.

La clausola `using index` consente di specificare i parametri di memorizzazione e la locazione del tablespace per un indice creato da un vincolo. Nel seguente esempio viene creata una chiave primaria sulla colonna `Societa` della tabella `AZIONE`. Al vincolo di chiave primaria viene assegnato il nome `PK_STOCK`. L'indice associato con la chiave primaria è inviato nel tablespace `INDICI`, con determinati parametri di memorizzazione. Sulla colonna `Simbolo` viene creato un vincolo unico e anche il suo indice viene posto nel tablespace `INDICI`. Entrambi i vincoli sono specificati a livello di tabella (invece che a livello di colonna) per illustrare meglio la sintassi di `using index`.

```
create table AZIONE (
    Societa      VARCHAR2(20),
    Simbolo      VARCHAR2(6),
    Industria   VARCHAR2(15),
    ChiusIeri   NUMBER(6,2),
    ChiusOggi   NUMBER(6,2),
    Volume       NUMBER,
    constraint PK_STOCK primary key (Societa)
        using index tablespace INDICI
        storage (initial 20K next 20K),
    constraint UQ_STOCK unique (Simbolo)
        using index tablespace INDICI
        storage (initial 20K next 20K))
;
```

Per ulteriori opzioni relative alla clausola `using index` si può consultare il paragrafo dedicato ai vincoli di integrità nella guida alfabetica di riferimento del Capitolo 37. Per le opzioni relative alle prestazioni nella creazione di indici si può consultare il paragrafo dedicato a `create index`.

Ricostruzione di un indice

ORACLE fornisce una funzionalità di ricostruzione veloce di indici che consente di creare di nuovo un indice senza rimuovere quello esistente. L'indice disponibile attualmente è utilizzato come fonte di dati per il nuovo indice, al posto della tabella. Durante la ricostruzione dell'indice è possibile cambiare i parametri di memorizzazione e l'assegnamento del tablespace.

Nel seguente esempio viene ricostruito l'indice `LAVORATORE_PK` (attraverso la clausola `rebuild`). I parametri di memorizzazione sono cambiati per poter utilizzare una dimensione di estensione iniziale di 10 MB e una dimensione di estensione successiva di 5 MB, nel tablespace `INDX_2`.

```
alter index LAVORATORE_PK rebuild
storage (initial 10M next 5M pctincrease 0)
tablespace INDX_2;
```

NOTA Quando l'indice `LAVORATORE_PK` viene ricostruito, deve esistere spazio sufficiente per entrambi gli indici, quello vecchio e quello nuovo. Dopo che il nuovo indice è stato creato, il vecchio viene rimosso e il suo spazio liberato.

Quando si crea un indice basato su una colonna inclusa precedentemente in un indice, ORACLE può essere in grado di utilizzare gli indici esistenti come fonte di dati per quello nuovo. Ad esempio, se si crea un indice su due colonne (`Nome, Alleggio`) e poi si decide di creare un indice soltanto sulla colonna `Nome`, ORACLE utilizza l'indice esistente come fonte di dati per il nuovo indice. Come risultato, le prestazioni del comando `create index` migliorano, se si crea l'indice in modo che possa trarre vantaggio da questa caratteristica.

19.2 Tablespace e struttura del database

Tutti conoscono il concetto di *file*; si tratta di un'area sul disco dove viene salvata l'informazione, dotata di un nome. La sua dimensione di solito non è fissa: se si aggiungono 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 dell'informazione nel file su varie sezioni più piccole del disco che non sono fisicamente vicine. Il sistema operativo gestisce la 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 nel suo schema organizzativo, ma la sua struttura va al di là del concetto di file. Un tablespace è un'area del disco costituita da uno o più file e può contenere molte tabelle, indici o cluster. Poiché un tablespace ha una dimensione fissa, quando si aggiungono righe alle sue tabelle può diventare pieno. Quando accade questo, il tablespace può essere ampliato da qualcuno che ha autorità di DBA. L'espansione è realizzata creando un nuovo file su disco e aggiungendolo al tablespace o estendendo i file di dati (chiamati anche *datafile*) esistenti. Le nuove righe possono essere aggiunte alle tabelle esistenti, che perciò avranno righe in entrambi i file. Uno o più tablespace insieme costituiscono un database.

Ogni tabella ha una singola area di spazio su disco, chiamata *segmento* e a essa riservata nel tablespace. Ogni segmento, a sua volta, ha un'area iniziale di spazio su disco, riservata a esso nel tablespace e chiamata *estensione iniziale*. Quando un segmento ha utilizzato tutto lo spazio, gli viene attribuita l'estensione successiva, un'altra singola area di spazio su disco. Quando ha utilizzato anche questa, gliene viene assegnata un'altra. Il processo continua con ogni tabella finché l'intero tablespace non è pieno. A questo punto occorre aggiungere un nuovo file al tablespace o estendere i file prima di poter ampliare le tabelle.

Se tutto ciò può risultare confuso, un vecchio parallelismo (Figura 19.2) può aiutare a chiarire le cose. Si può pensare a un database come a un'area coltivabile comunale, dove ogni tablespace è un lotto recintato.

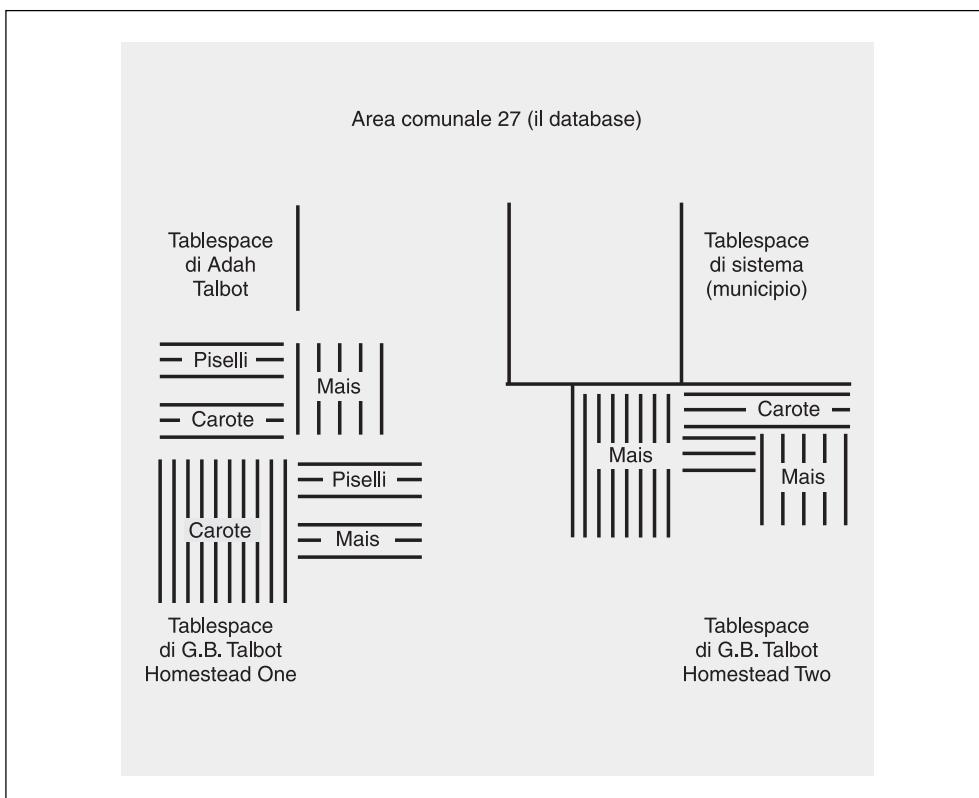


Figura 19.2 Una struttura di un database ORACLE paragonata a un'area coltivabile comunale.

All'inizio l'area è piuttosto rada, vi sono solo alcuni lotti recintati ed è piena di spazi vuoti, non rivendicati. Il lotto uno (il tablespace uno) è posseduto da G.B. Talbot, che l'ha chiamato "Homestead One". Egli pianta varie zone dell'orto: file di mais, di piselli, di carote. Ognuna di queste corrisponde a una tabella con le sue righe e ciascuna è piantata nella sua zona o "estensione iniziale". Dopo le carote decide di piantare altro mais, ma ha terminato questa sezione del lotto, perciò lo pianta dall'altra parte del recinto. Il suo mais ora è nell'estensione successiva (la sua seconda estensione). Egli continua allo stesso modo con file di piselli e carote, finché la zona diventa una sorta di mosaico di sezioni.

Alla fine riempie l'intero lotto. Il suo tablespace è pieno, quindi acquista del terreno vuoto dall'altra parte del blocco, recinta il lotto e lo chiama "Homestead Two". Il suo tablespace è ora più ampio. Anche se i lotti (i file) non sono fisicamente connessi, Talbot li possiede entrambi. Il tablespace è ancora "G.B. Talbot" e contiene due lotti "Homestead One" e "Homestead two". Ora Talbot può piantare altre file di mais, piselli e carote nel suo secondo lotto, continuando a consumare "estensioni" aggiuntive.

Ogni database contiene anche un tablespace di sistema. Questo è il palazzo comunale, dove vengono conservati gli indirizzi e le registrazioni di proprietà. Esso contiene il dizionario di dati e i nomi e le posizioni di tutti i tablespace, tabelle, indici e cluster per il database.

La Figura 19.3 mostra come appare un database con una tipica collezione di tablespace, tabelle, indici, cluster ed estensioni. Il database viene creato e denominato dall'amministratore del database, che di solito imposta i tablespace e ne concede l'utilizzo a singoli utenti. In questo database, chiamato TALBOT, sono stati costruiti i tablespace ADAH, JONES, SYSTEM e GEORGE.

In un tablespace vi sono tabelle, indici e cluster. A ogni tabella viene all'inizio allocata un'estensione iniziale e una dimensione di estensione successiva che stabilisce la quantità di spazio allocata ogni volta che termina l'estensione corrente. In questa figura, le tabelle COMFORT, CLIMA e LAVORATORE sono tutte cresciute al di là dell'estensione iniziale e hanno consumato completamente il file iniziale, HOME.ONE, del tablespace.

COMFORT si è espansa in due estensioni aggiuntive nel secondo file. Anche CLIMA e LAVORATORE hanno un'estensione in più nel secondo file. Un indice sulla tabella LAVORATORE è stato creato dopo che era diventato necessario il secondo file, come anche un cluster per la tabella HOCKEY (i cluster verranno trattati nel paragrafo "Cluster" alla fine di questo capitolo).

create tablespace

Il comando `create tablespace` consente di assegnare immediatamente al tablespace uno o più file; specifica anche uno spazio di default per ogni tabella creata senza che nell'istruzione `create table` venga menzionata alcuna esplicita clausola di memorizzazione.

Ecco la sintassi di base per il comando `create tablespace`:

```
create tablespace TALBOT datafile 'HOME.ONE' size 1000K
```

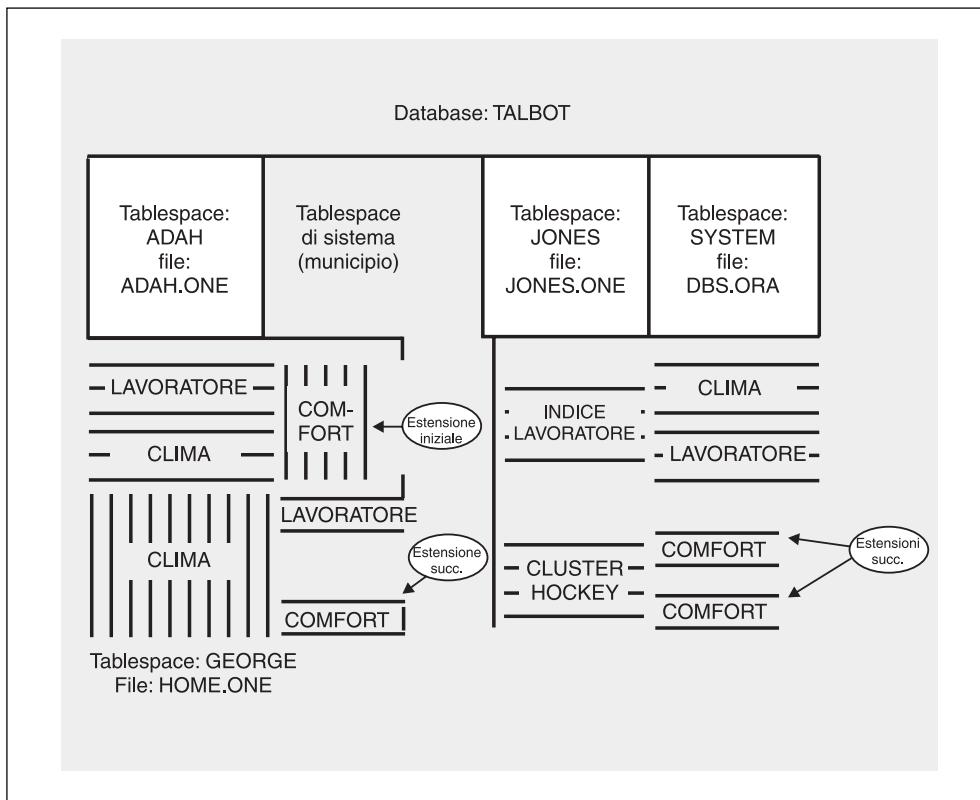


Figura 19.3 Una struttura di un database ORACLE descritta normalmente.

```
default storage (initial 25K next 10K
               minextents 1 maxextents 100
               pctincrease 0)
permanent;
```

NOTA I tablespaces possono essere designati come *READ ONLY* attraverso il comando *alter tablespace*, che è descritto nella guida alfabetica di riferimento del Capitolo 37.

NOTA La parola chiave *permanent* nel comando *create tablespace* comunica a ORACLE che si stanno memorizzando nel tablespace oggetti permanenti (come tabelle). Se il tablespace è utilizzato solo per segmenti temporanei, è possibile specificare la parola chiave *temporary*. Il valore di default è *permanent*.

Il file iniziale assegnato a questo tablespace è incluso nel comando insieme alla dimensione in byte, non in blocchi. Il numero di byte è un intero e può essere seguito da una lettera “K” (per moltiplicare per 1024, circa mille) o da una “M” (per moltiplicare per 1048576, circa un milione). *default storage* imposta lo spazio assegnato a una tabella se nell’istruzione *create table* non è specificato *storage*.

Qui l'estensione iniziale è di 25 Kbyte (non blocchi) e la successiva (incrementale) è di 10 Kbyte.

`minextents` consente di riservare, quando viene creata una tabella, estensioni aggiuntive oltre la prima. Queste non devono essere necessariamente contigue fisicamente con l'estensione iniziale, o fra di loro, ma almeno lo spazio è riservato.

`maxextents` è il limite di estensioni aggiuntive consentite. A partire da ORACLE7.3 è possibile specificare `maxextents UNLIMITED`, nel qual caso non vi è alcun limite al numero di estensioni consentite per la tabella o indice.

`pctincrease` è un fattore di crescita per le estensioni. Quando viene impostato a un valore diverso da zero, ogni estensione incrementale sarà più grande rispetto alla precedente del valore percentuale specificato. Questo provoca una crescita esponenziale dello spazio allocato per la tabella. Se il volume dei dati nella tabella cresce con una velocità costante, conviene impostare `pctincrease` a 0.

I valori di default per la memorizzazione dipendono dal sistema operativo. I valori minimo e massimo per ognuna di queste opzioni sono descritti nella guida alfabetica di riferimento del Capitolo 37, nei paragrafi dedicati alla memorizzazione e a `create table`. Queste opzioni possono essere modificate con il comando `alter tablespace`. Il comando `create table` per la tabella REGISTRO è il seguente:

```
create table REGISTRO (
  DataAzione      DATE,
  Azione          VARCHAR2(8),
  Articol         VARCHAR2(30),
  Quantita        NUMBER,
  TipoQuantita   VARCHAR2(10),
  Tasso           NUMBER,
  Importo         NUMBER(9,2),
  Persona         VARCHAR2(25)
)
tablespace TALBOT
;
```

In questa forma la tabella eredita le definizioni di memorizzazione di default del tablespace TALBOT. Per ridefinirle occorre utilizzare la clausola `storage` nel comando `create table`:

```
create table REGISTRO (
  DataAzione      DATE,
  Azione          VARCHAR2(8),
  Articol         VARCHAR2(30),
  Quantita        NUMBER,
  TipoQuantita   VARCHAR2(10),
  Tasso           NUMBER,
  Importo         NUMBER(9,2),
  Persona         VARCHAR2(25)
)
tablespace TALBOT
storage (initial 5K next 5K
          minextents 2 maxextents 50
          pctincrease 0)
;
```

19.3 Cluster

La creazione di cluster è un metodo per salvare tabelle che sono strettamente correlate e spesso unite nella stessa area sul disco. Ad esempio, invece di porre la tabella LAVORATORE in una sezione del disco e la tabella COMPITOLAVORATORE 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 di solito unite in una query (ad esempio Nome per le tabelle LAVORATORE e COMPITOLAVORATORE). 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 dei nomi per le tabelle, *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 può coincidere con quello di una delle colonne di una tabella che si vuole inserire nel cluster o può essere qualsiasi altro nome valido. Ecco un esempio:

```
create cluster LAVORATOREeCOMPITO ( Judy VARCHAR2(25));
```

Cluster created.

Questo codice crea un cluster (uno spazio riservato, come avverrebbe per una tabella) che non contiene nulla. L'utilizzo di Judy come nome per la chiave di cluster è irrilevante; infatti, non verrà mai utilizzato di nuovo. Poi vengono create delle tabelle da includere nel cluster:

```
create table LAVORATORE (
  Nome      VARCHAR2(25) not null,
  Eta       NUMBER,
  Alloggio  VARCHAR2(15)
)
cluster LAVORATOREeCOMPITO (Nome)
;
```

Prima di inserire delle righe in LAVORATORE, si deve creare un indice cluster:

```
create index LAVORATOREECOMPITO_NDX
on cluster LAVORATOREeCOMPITO;
```

Si ricordi che la presenza della clausola cluster preclude l'utilizzo delle clausole tablespace o storage. È opportuno notare come questa struttura sia diversa dall'istruzione create table standard:

```
create table LAVORATORE (
  Nome      VARCHAR2(25) not null,
  Eta       NUMBER,
  Alloggio  VARCHAR2(15)
);
```

Nella prima istruzione create table la clausola cluster LAVORATOREeCOMPITO (Nome) segue la chiusura delle parentesi che includono l'elenco di colonne da creare nella tabella.

LAVORATOREeCOMPITO è il nome del cluster creato precedentemente. Nome è la colonna di questa tabella che viene salvata nella chiave di cluster Judy. 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 viene specificato esplicitamente che la colonna Nome deve andare nella chiave di cluster Judy. L'accoppiamento è realizzato solo in base alla posizione: Nome e Judy sono 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 COMPITOLAVORATORE (
  Nome      VARCHAR2(25) not null,
  Compito   VARCHAR2(25) not null,
  Capacita  VARCHAR2(15)
)
cluster LAVORATOREeCOMPITO (Nome)
; 
```

Memorizzazione delle tabelle

La Figura 19.4 illustra come sono memorizzati i dati in un cluster. Si ricordi che la tabella LAVORATORE ha tre colonne: Nome, Eta e Alloggio. Anche la tabella COMPITOLAVORATORE ha tre colonne: Nome, Compito e Capacita. Quando queste due tabelle vengono unite in un cluster, ogni singolo Nome è in realtà memorizzato solo una volta nella chiave di cluster. A ogni Nome sono 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, il *cluster di hash*, utilizza i valori delle colonne di cluster per determinare la locazione fisica in cui è memorizzata la riga. Per maggiori informazioni si può consultare la voce create cluster nella guida alfabetica di riferimento del Capitolo 37.

19.4 Sequenze

ORACLE ha risolto il vecchio problema di assegnare numeri unici, come gli identificativi di clienti, senza dover creare una tabella speciale e gestire gli aggiornamenti e le coincidenze. Questo si ottiene utilizzando il comando create table, come mostrato qui:

```
create sequence IDCliente increment by 1 start with 1000;
```

| ETA ALLOGGIO | NOME | COMPITO | CAPACITA |
|--------------|----------------------------|-------------|--------------|
| 23 | PAPA KING ADAH TALBOT | OPERAIO | BUONO |
| 29 | ROSE HILL ANDREW DYE | | |
| 22 | CRANMER BART SARJEANT | | |
| 18 | ROSE HILL DICK JONES | FABBRO | ECCELLENTE |
| 16 | MATTS DONALD ROLLO | | |
| 43 | WEITBROCHT ELBERT TALBOT | ARATORE | LENTO |
| 41 | ROSE HILL GEORGE OSCAR | | |
| 55 | PAPA KING GERHARDT KENTGEN | | |
| 15 | HELEN BRANDT | TREBBIATORE | MOLTO VELOCE |
| 33 | MATTS JED HOPKINS | | |
| 27 | ROSE HILL JOHN PEARSON | TREBBIATORE | |
| | | TAGLIALEGNA | BUONO |
| | | FABBRO | MEDIO |
| ROSE HILL | KAY AND PALMER WALLBOM | | |
| 21 | ROSE HILL PAT LAVAY | | |
| 25 | CRANMER PETER LAWSON | | |
| WEITBROCHT | RICHARD KOCH AND BROTHERS | | |
| 35 | MATTS ROLAND BRANDT | | |
| 32 | MULLERS VICTORIA LYNN | FABBRO | PRECISO |
| 67 | WILFRED LOWELL | OPERAIO | MEDIO |
| | | ARATORE | MEDIO |
| 15 | CRANMER WILLIAM SWING | | |

Dalla tabella LAVORATORE

Dalla tabella COMPITOLAVORATORE

Chiave di cluster

Figura 19.4 Memorizzazione dei dati nei cluster.

Questo codice crea una sequenza a cui si può accedere attraverso le istruzioni insert e update (anche select, ma è alquanto raro). Tipicamente, la sequenza in questione viene creata 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 successivo numero di sequenza disponibile per la sequenza IDCliente. Questo numero è unico; ORACLE non lo assegna a nessun altro. Per impiegare lo stesso numero più di una volta (come in una serie di inserimenti nelle relative tabelle) viene impiegato, dopo

il primo utilizzo, CurrVal invece di NextVal. L'utilizzo di NextVal assicura che la tabella della sequenza venga incrementata e che il numero ottenuto sia unico. Una volta che si è utilizzato NextVal, quel numero viene salvato in CurrVal per impiegarlo altrove, finché non si utilizza di nuovo NextVal e allora sia NextVal che CurrVal cambiano assumendo il valore del nuovo numero della sequenza.

Se si utilizza sia NextVal che CurrVal in una singola istruzione SQL, entrambi contengono il valore restituito da NextVal. Nessuno di questi può essere impiegato in una sottoquery, come colonne 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 e si può far ripartire il ciclo della sequenza dal valore iniziale quando viene raggiunto il massimo. Per maggiori informazioni si può consultare create sequence nella guida alfabetica di riferimento (Capitolo 37).

- Capitolo 20
- **SQLPLUS**
-
- 20.1 **Generazione di codice per una query**
- 20.2 **Caricamento di variabili**
- 20.3 **Creazione e annidamento di file di avvio e comandi**
- 20.4 **Riepilogo**

SQLPLUS possiede un certo numero di caratteristiche che vanno molto al di là dei report e dell'ambiente SQL interattivo e ne fanno un effettivo generatore di codice. Infatti può creare dinamicamente codice SQL e SQL-PLUS ed eseguirlo. Questo capitolo esplora alcune tecniche per sfruttare al meglio l'utilizzo di SQLPLUS.

Una delle reali potenzialità di SQL è la capacità di favorire esami molto più approfonditi dei dati. Poiché le query possono essere formulate, eseguite e riformulate molto velocemente, da interrogazioni elementari si può arrivare a problemi molto più avanzati.

20.1 **Generazione di codice per una query**

Ad esempio, si supponga di voler sapere quante tabelle vi sono correntemente nel proprio nome utente. Si potrebbe utilizzare questa query:

```
select Table_Name from USER_TABLES;
```

USER_TABLES contiene informazioni sulle tabelle e Table_Name è la colonna che contiene i nomi delle tabelle stesse. Il risultato della query è il seguente:

| TABLE_NAME |
|-------------------|
| ----- |
| GENERAZIONE |
| REGISTRO |
| ALLOGGIO |
| COMPITO |
| LAVORATORE |
| COMPITOLAVORATORE |

6 rows selected.

Ora si conoscono i nomi delle proprie tabelle, ma le dimensioni? Quante righe contengono? Si potrebbe scrivere una query come quella seguente, per ognuna di queste tabelle:

```
select COUNT(*) from GENERAZIONE;
```

In questo modo si otterrebbero le risposte necessarie; si potrebbe costruire un file di avvio che contenga sei query come questa e poi eseguirlo regolarmente per registrare la crescita delle varie tabelle nel database. E se venisse aggiunta un'altra tabella? Naturalmente sarebbe necessario ritornare indietro e aggiungere la nuova tabella al file di avvio. Si potrebbe anche creare un file di avvio denominato master.sql come quello seguente:

```
rem master.sql - crea ed esegue slave.sql

set feedback off
set heading off
spool slave.sql
select 'select COUNT(*) from'||Table_Name||';' from
USER_TABLES;
spool off
```

L'esecuzione produce un file chiamato slave.sql che contiene quanto segue:

```
select COUNT(*) from GENERAZIONE;
select COUNT(*) from REGISTRO;
select COUNT(*) from ALLOGGIO;
select COUNT(*) from COMPITO;
select COUNT(*) from LAVORATORE;
select COUNT(*) from COMPITOLAVORATORE;
```

L'esecuzione di slave.sql produce i conteggi delle righe per ciascuna di queste tabelle. Il vantaggio di questo approccio è che crea sempre tante istruzioni select quante sono le tabelle elencate in USER_TABLES. Il trucco consiste nel fare in modo che select contenga una colonna letterale che è anch'essa un'istruzione select. Poiché i risultati vengono registrati su un file, si ottiene un codice eseguibile, in questo caso una serie completa di istruzioni select che contano le righe in ogni tabella.

set feedback off evita che un messaggio del tipo ‘6 row selected’ (6 colonne selezionate) appaia in slave.sql e set heading off fa la stessa cosa per le intestazioni delle colonne. I messaggi “row selected” e le intestazioni delle colonne non sono istruzioni SQL o SQLPLUS, perciò non devono essere presenti in slave.sql.

Si può estendere questa tecnica di base in vari modi. Ad esempio, si può rendere automatica l'esecuzione di slave.sql includendolo nel file di avvio originario master.sql e registrando l'output in un altro file, table.lst:

```
rem master.sql - crea ed esegue slave.sql

set feedback off
set heading off

spool slave.sql
select 'select COUNT(*) from'||Table_Name||';' from
USER_TABLES;
```

```
spool off
```

```
spool table.lst
start slave.sql
spool off
```

Naturalmente il file table.lst conterrà soltanto una colonna di numeri, le righe contate per ognuna delle tabelle. Per includere anche i nomi delle tabelle è necessaria una query leggermente più complicata:

```
set feedback off
set heading off

spool slave.sql
select 'select '||'''||Table_Name||'''||', COUNT(*) from '||'
      Table_Name||';' from USER_TABLES;
spool off

spool table.lst
start slave.sql
spool off
```

Questo produce un file slave.sql come il seguente:

```
select 'GENERAZIONE', COUNT(*) from GENERAZIONE;
select 'REGISTRO', COUNT(*) from REGISTRO;
select 'ALLOGGIO', COUNT(*) from ALLOGGIO;
select 'COMPITO', COUNT(*) from COMPITO;
select 'LAVORATORE', COUNT(*) from LAVORATORE;
select 'COMPITOLAVORATORE', COUNT(*) from COMPITOLAVORATORE;
```

La sola modifica è l'inclusione dei nomi delle tabelle come letterali, racchiusi tra apici, a sinistra di COUNT(*), ottenuta con il codice seguente:

```
||'''||
```

I quattro apici singoli producono un unico apice nel risultato perché i due apici centrali sono interpretati da ORACLE come un apice singolo letterale, invece che come un delimitatore SQL. I due esterni non sono diversi dagli apici singoli che racchiudono 'select'. L'effetto finale è la concatenazione di un apice tra 'select' e Table_Name. Lo stesso avviene a destra di Table_Name.

Sapendo che ORACLE interpreta due apici singoli adiacenti come un unico apice, si sarebbe potuto scrivere il codice seguente:

```
select 'select '''||Table_Name||'''', COUNT(*) from '||'
      Table_Name||';' from USER_TABLES;
```

Però, quando la creazione di file di avvio che generano codice diventa più complessa, anche la gestione di apici singoli letterali diviene più complicata. Si possono avere istanze di select inserite in select inserite in altri select, dove potrebbe essere necessaria una lunga stringa di apici singoli per produrre il risultato desiderato. Alla lunga diventa quasi impossibile calcolare quanti apici singoli consecutivi sono necessari. Utilizzando sempre ||"||" dovunque è necessario un apice singolo letterale, è più facile la gestione di stringhe anche complesse.

L'esecuzione di slave.sql produce ora un file denominato table.lst, simile al seguente:

| | |
|-------------------|-----|
| GENERAZIONE | 16 |
| REGISTRO | 225 |
| ALLOGGIO | 6 |
| COMPITO | 6 |
| LAVORATORE | 19 |
| COMPITOLAVORATORE | 10 |

In molti casi master.sql o slave.sql potrebbero contenere comandi SQLPLUS per formattare le colonne di report.lst con istruzioni, come ttitle e btitle, inserite dopo che slave.sql è stato creato, ma prima che venisse eseguito, nel modo seguente:

```

set feedback off
set heading off

spool slave.sql
select 'select ''||''''||Table_Name||'''', COUNT(*) from '||
      Table_Name||';' from USER_TABLES;
spool off

ttitle 'Elenco di tabelle e dimensioni'
set heading on
set feedback 1
spool table.lst
start slave.sql
spool off

```

Questo metodo però non funziona, perché ttitle inserisce un titolo superiore per ogni istruzione select che viene eseguita e slave.sql ne ha sei. In table.lst vi sarebbe un ttitle e le intestazioni di colonna sopra ogni riga. Inoltre verrebbe restituito un messaggio '1 row selected' per ogni riga. L'aspetto del report sarebbe assai confuso.

Si può generare un report con il numero di righe di ogni tabella attraverso una vista accuratamente costruita dalla tabella USER_TABLES, eventualmente utilizzando un'unione delle tabelle, ma attualmente SQL non ha i mezzi per ottenere una vista con un numero dinamico di istruzioni select unite. Tuttavia, questo metodo può essere approssimato. Il primo tentativo è mostrato nella Figura 20.1 e il risultato nella Figura 20.2.

Al punto 1 si impostano pagesize e linesize in modo che siano abbastanza grandi da evitare problemi di formattazione. Se linesize è troppo piccolo, alcune lunghe istruzioni SQL generate potrebbero rimanere a metà. Se pagesize è troppo piccolo, viene inserita una riga vuota ogni volta che le istruzioni generate superano la lunghezza della pagina. Ovvero, se slave.sql avesse istruzioni SQL di 200 linee (cosa non insolita) e pagesize fosse impostato a 50, nell'istruzione SQL vi sarebbero tre righe vuote, una ogni 50 righe. Questo farebbe fallire l'esecuzione dell'istruzione. Però, pagesize viene impostato a un valore molto grande.

```

rem master.sql - crea ed esegue slave.sql

set pagesize 30000 ┌─────────┐ 1
set linesize 200   └─────────┘

set timing off
set time off
set feedback off
set heading off
set echo off
ttitle off
btitle off
spool slave.sql ┌─────────┐ 2

prompt create or replace view USER_TABLE_SIZE as
select DECODE(RowNum,1,null,'union') ||
'select ''||''''||Table_Name||'''||' Name'||'
', COUNT(*) Row_Count from '||Table_Name
      from USER_TABLES; ┌─────────┐ 3
prompt /           4
spool off          5
    6
    7

```

Figura 20.1 Il primo tentativo di creare una vista delle dimensioni delle tabelle.

```

create or replace view USER_TABLE_SIZE as A
select 'GENERAZIONE' Name, COUNT(*) Row_Count from GENERAZIONE
union select 'REGISTRO' Name, COUNT(*) Row_Count from REGISTRO
union select 'ALLOGGIO' Name, COUNT(*) Row_Count from ALLOGGIO
union select 'COMPITO' Name, COUNT(*) Row_Count from COMPITO
union select 'LAVORATORE' Name, COUNT(*) Row_Count from LAVORATORE
union select 'COMPITOLAVORATORE' Name, COUNT(*) Row_Count from
COMPITOLAVORATORE
/

```

Figura 20.2 Il risultato del primo tentativo (contenuto di slave.sql).

Al punto 2 ogni comando SQLPLUS che potrebbe produrre messaggi, titoli, intestazioni, feedback o qualsiasi altro testo non eseguibile viene disattivato in modo da assicurare che il codice generato in slave.sql sia eseguibile. In alternativa, invece di porre questo elenco in ogni file di avvio si possono inserire queste sette righe in un piccolo file di avvio indipendente, chiamato ad esempio off.sql. Quindi le righe possono essere sostituite dalla singola riga seguente (quando si utilizza il comando start un file di avvio con estensione .sql può essere abbreviato al solo nome del file):

```
start off
```

Si potrebbe utilizzare una tecnica simile alla fine del file master.sql per riattivare le caratteristiche SQLPLUS che di solito si preferisce avere in funzione, come set feedback on e set heading on.

Al punto 3 viene utilizzato il comando prompt per “scrivere” l’istruzione create or replace view nel file di registrazione. Il suo effetto si può vedere al punto A della Figura 20.2.

Ora le cose cominciano a complicarsi, almeno a prima vista. Il punto 4 contiene un’istruzione DECODE:

```
select DECODE(RowNum,1,null,'union ')||
```

Quando viene eseguita questa istruzione select, la prima riga recuperata restituisce un RowNum di 1, quindi DECODE produce un risultato NULL, ovvero niente. L’effetto può essere visto al punto B. Per ogni riga dopo questa, RowNum non è 1, quindi DECODE produce la parola union. L’effetto è mostrato al punto C nella Figura 20.2. Ogni riga, eccetto la prima, inizia con la parola union.

Il punto 5 è simile al metodo mostrato precedentemente, tranne che alla colonna Table_Name (che conterrà i nomi effettivi delle tabelle) è stato assegnato l’alias Name:

```
'select '||'||'||Table_Name||'||'||' Name'||
```

Name sarà il nome della colonna nella vista USER_TABLE_SIZE. Si possono vedere gli effetti in tutte le righe che contengono un select dal punto B in avanti. Al punto 6, allo stesso modo, COUNT(*) viene rinominata come Row_Count.

Al punto 7 viene utilizzato di nuovo il comando prompt per produrre un terminatore SQL.

Questo tentativo va quasi bene. L’istruzione create or replace view viene ignorata a causa della riga vuota che la segue. Per correggere il problema, il comando prompt per create or replace view è sostituito da un’aggiunta a DECODE:

```
select DECODE(RowNum,1,'create or replace view  
USER_TABLE_SIZE as  
'||'union ')||  
'select '||'||'||Table_Name||'||'||' Name'||  
, count(*) Row_Count from '|||Table_Name  
from USER_TABLES;  
prompt /
```

Ora slave.sql contiene il seguente codice:

```
create or replace view USER_TABLE_SIZE as  
select 'GENERAZIONE' Name, count(*) Row_Count from GENERAZIONE  
union select 'REGISTRO' Name, count(*) Row_Count from REGISTRO  
union select 'ALLOGGIO' Name, count(*) Row_Count from ALLOGGIO  
union select 'COMPITO' Name, count(*) Row_Count from COMPITO  
union select 'LAVORATORE' Name, count(*) Row_Count from LAVORATORE  
union select 'COMPITOLAVORATORE' Name, count(*) Row_Count from  
COMPITOLAVORATORE  
/
```

Dopo che slave.sql è stato eseguito, la nuova vista esiste ed è possibile effettuarvi delle query:

```
select * from USER_TABLE_SIZE;
```

| NAME | Row_Count |
|-------------------|-----------|
| GENERAZIONE | 16 |
| REGISTRO | 225 |
| ALLOGGIO | 6 |
| COMPITO | 6 |
| LAVORATORE | 19 |
| COMPITOLAVORATORE | 10 |

6 rows selected.

Naturalmente questa istruzione select e probabilmente alcuni titoli e altri comandi di formattazione potrebbero essere inclusi tutti nel file di avvio master.sql, dopo la creazione e l'esecuzione di slave.sql. In seguito, l'esecuzione di master.sql produrrà sempre un elenco corrente delle tabelle e delle loro dimensioni.

20.2 Caricamento di variabili

Il metodo appena descritto di registrazione su un file e di creazione di codice da istruzioni SQL e comandi prompt può essere utilizzato anche per caricare le variabili in SQLPLUS in modo da utilizzarle in successive istruzioni SQL, ttitle o btitle, prompt e simili. Ad esempio, si supponga di dover eseguire una serie di report che effettuano tutti calcoli utilizzando SUM e AVG sulla colonna Importo nella tabella REGISTRO. Il metodo illustrato nei Capitoli 10 e 16 consisteva nel creare un vista che conteneva questi calcoli e poi unire la vista ad altre tabelle nell'istruzione select. Nella maggior parte dei casi questo approccio è appropriato.

Si supponga però che la tabella su cui devono essere calcolate SUM e AVG sia molto grande. Per ogni istruzione SQL che deve essere eseguita, la vista ricalcola SUM e AVG per l'intera tabella. Se queste funzioni potessero essere calcolate soltanto una volta e poi conservate in una variabile, questa variabile potrebbe poi essere utilizzata nelle successive istruzioni senza elaborare di nuovo la vista. Questo di solito non è un approccio sensato se la tabella varia frequentemente mentre vengono prodotti i report, ma se i report vengono eseguiti di notte o se i dati sono stabili, funziona bene (si può anche utilizzare un lock table in row share o share update mode). Di seguito è riportato un esempio di caricamento di due variabili con dati ricavati da una query:

```
set pagesize 32000
set linesize 200

start off

column Piegalo newline
```

```
spool slave.sql
select 'define Media =', AVG(Amount),
       'define Totale    =' PiegaLo, SUM(Importo)
  from REGISTRO;
spool off
```

```
start slave
```

Questo listato produce un file di avvio slave.sql, simile a quello seguente:

```
define Media = 4.57075556
define Totale   = 1028.42
```

Quando slave.sql viene avviato, queste variabili sono definite in SQLPLUS e possono essere utilizzate nelle successive istruzioni SQL. Il file off.sql assicura che il file slave.sql prodotto sia eseguibile. L'utilizzo di questa tecnica con una sola variabile è semplice, mentre la definizione di due variabili contemporaneamente richiede maggiore lavoro. SQLPLUS non accetta questo:

```
define Media = 4.57075556 define Totale   = 1028.42
```

perché sono presenti due define in una sola riga. Perciò alla colonna letterale 'define Total=' viene assegnato un alias PiegaLo e prima viene inserito un comando di colonna column PiegaLo newline. Così questa colonna viene scritta in una nuova riga. Se si hanno più colonne, si può utilizzare lo stesso alias per ogni riga in cui è necessario andare a capo.

Il singolo comando di colonna potrebbe poi controllarli tutti. Per inciso, newline ha lo stesso effetto di fold_before 1, trattato nel Capitolo 13.

Modifica dei valori delle variabili

Un variabile, una volta definita, può subire modifiche. Qui, solo come esempio, la media viene sommata a se stessa (dopo che è già stata definita una volta):

```
column B format 999999999.9999
spool slave.sql
select 'define Media =', (AVG(Importo) + &Media)  B,
       'define Totale    =', SUM(Importo)
  from REGISTRO;
spool off
start slave
```

Il codice precedente produce quanto segue:

```
define Media      =      9.1415
define Totale    = 1028.42
```

Si noti come l'utilizzo di column B format 999999999.9999 influisca sul modo in cui appare ora il valore di Media; è opportuno confrontarlo con quello precedente, quando gli era stato applicato soltanto set numwidth 10.

Utilizzo di variabili con ttitle

Dal momento che qualsiasi variabile può essere utilizzata anche in ttitle o btitle (per maggiori dettagli si può consultare il Capitolo 13), si potrebbe creare un'istruzione SQL che carichi un valore in una variabile e lo utilizzi nel comando ttitle. Questo si dimostrerebbe particolarmente utile quando il valore che si desidera inserire in ttitle non può essere ottenuto dall'istruzione select principale del report per cui ttitle è l'intestazione.

Stringhe di caratteri e date

Quando si carica una variabile in questo modo, le stringhe di caratteri e le date devono essere concatenate con apici singoli, come mostrato di seguito:

```
select 'define Citta = ''||'||Citta||''''
from COMFORT
where Mezzogiorno is NULL;
```

questo produce quanto segue:

```
define Citta = 'San Francisco'
```

Quando la variabile viene poi utilizzata in una successiva istruzione SQL, non è racchiusa fra apici singoli:

```
select Mezzanotte
from COMFORT
where Citta = &Citta;
```

Se si sbaglia a concatenare la stringa, quando la prima istruzione select viene eseguita define riceve solo la prima parola di una stringa di testo costituita da più parole (come San Francisco). Durante la creazione della variabile, concatenando le date e le stringhe di testo e lasciando i numeri così come sono, tutte le variabili appaiono allo stesso modo, ovvero non hanno apici nelle successive istruzioni select.

20.3 Creazione e annidamento di file di avvio e comandi

La Figura 20.3 mostra un file di avvio, master.sql, utilizzato per generare un semplice report su tutte le tabelle nel database. Il suo scopo è illustrare alcune tecniche aggiuntive che possono essere impiegate per la generazione di codice in file di avvio.

Il punto 1 mostra come si può scrivere in un file di avvio slave.sql un comando host o di altro tipo. Si ricordi che \$ è equivalente alla parola host ed è utilizzato per eseguire un modulo chiamato cls sul sistema operativo host (in questo caso si tratta di un comando che azzerà lo schermo). Poiché questo è un prompt, \$cls viene semplicemente scritto nel file slave.sql, come anche le parole spool table.lst. Gli effetti possono essere visti al punto A nella Figura 20.4.

```

rem master.sql - crea ed esegue slave.sql

set pagesize 32000
set linesize 200

start off

column PiegaLo newline

spool slave.sql

prompt $cls
prompt spool table.lst 1
prompt prompt Inizio report tabelle 2
prompt prompt

select 'define Table  = '||'||'||Table_Name||'||', 3
      'prompt Working on '||'||'||Table_Name||'||', '
      'prompt
      ','
      'start sizing.sql'
  from USER_TABLES;

prompt prompt Report tabelle completati 4
prompt spool off
prompt $print table.lst

spool off

start slave

```

Figura 20.3 File di avvio e comandi annidati.

Il punto 2 della Figura 20.3 è un po' diverso. Qui un prompt scrive un altro prompt nel file slave. Quando viene eseguito slave.sql, i prompt vengono visualizzati sullo schermo, viene eseguito sizing.sql per ogni tabella e i risultati vengono registrati in table.lst.

Il punto 3 produce gli effetti mostrati al punto C e nelle righe seguenti: un gruppo di quattro comandi per ogni tabella. Quando viene eseguito slave.sql, i prompt vengono visualizzati sullo schermo, viene eseguito sizing.sql per ogni tabella e i risultati vengono registrati in table.lst.

Quando slave.sql è terminato, la riga finale invia alla stampante il file table.lst che è stato creato (punti 1 e D).

Il file di avvio sizing.sql (punti 3 e C) contiene il seguente codice:

```

describe &Table;
select 'Questa tabella contiene ||count(*)|| righe.' from &Table;
prompt
prompt
prompt

```

```

$cls
spool table.lst    A
prompt Inizio report tabelle B
prompt

define Table      = 'GENERAZIONE'
prompt Working on 'GENERAZIONE'
prompt
start sizing.sql

define Table      = 'REGISTRO'
prompt Working on 'REGISTRO'
prompt
start sizing.sql

define Table      = 'ALLOGGIO'
prompt Working on 'ALLOGGIO'
prompt
start sizing.sql

define Table      = 'COMPITO'
prompt Working on 'COMPITO'
prompt
start sizing.sql

define Table      = 'LAVORATORE'
prompt Working on 'LAVORATORE'
prompt
start sizing.sql

define Table      = 'COMPITOLAVORATORE'
prompt Working on 'COMPITOLAVORATORE'
prompt
start sizing.sql

prompt Report tabelle completati
spool off          C
$print table.lst    D

```

Figura 20.4 Gli effetti del file master.sql.

Si noti che sizing.sql si basa sulla variabile &Table che viene cambiata ogni volta appena prima che sizing.sql venga eseguito. Il risultato di tutti questi file, master.sql, slave.sql e sizing.sql, è il report mostrato nella Figura 20.5.

```
Inizio report tabelle
```

```
Working on 'GENERAZIONE'
```

| Name | Null? | Type |
|-------------|-------|--------------|
| FIGLIO | | VARCHAR2(10) |
| SEX | | CHAR(1) |
| MUCCA | | VARCHAR2(10) |
| TORO | | VARCHAR2(10) |
| DATANASCITA | | DATE |

Questa tabella contiene 16 righe.

```
Working on 'REGISTRO'
```

| Name | Null? | Type |
|--------------|-------|--------------|
| DATAAZIONE | | DATE |
| AZIONE | | VARCHAR2(8) |
| ARTICOLO | | VARCHAR2(30) |
| QUANTITA | | NUMBER |
| TIPOQUANTITA | | VARCHAR2(10) |
| TASSO | | NUMBER |
| IMPORTO | | NUMBER(9,2) |
| PERSONA | | VARCHAR2(25) |

Questa tabella contiene 225 righe.

.

.

.

(E così via, per ogni tabella. Le altre non sono mostrate)

.

.

.

Report tabelle completato

Figura 20.5 Report prodotto da slave.sql, che richiama ripetutamente sizing.sql.

Utilizzo di processi host

Nella produzione di questo report sono stati utilizzati due processi host. Uno ha azzerato lo schermo e l'altro ha stampato il report una volta completato. Effettivamente possono anche essere utilizzate operazioni host più complesse, ad esempio per riformattare i risultati di una query o per cancellare i file di lavoro creati. È anche possibile utilizzare SQLPLUS per creare macro da utilizzare in un dato editor, poi richiamare l'editor, eseguire le macro e ritornare a SQLPLUS.

Una robusta funzionalità per query interattive in lingua naturale è stata costruita utilizzando SQLPLUS e un linguaggio host interpretativo, dove ognuno scrive ripetutamente codice per l'altro e lo richiama, supportando così interazioni con l'utente.

20.4 Riepilogo

Le tecniche mostrate in questo capitolo sono state utilizzate per molti complessi obiettivi di realizzazione di report e manipolazione di dati, dalla creazione del report di una tabella che calcola la dimensione media delle righe (costruendo dinamicamente un select con VSIZE per ogni colonna di ogni tabella) a un report che mostra su quali tabelle agisce ogni applicazione e se nelle tabelle sono state selezionate, inserite, aggiornate o cancellate righe per ogni blocco. Vi sono numerose altre possibilità: convalida degli indici, creazione, copia o scaricamento di tabelle; persino creazione di file batch dinamici che possono uscire da SQLPLUS (se necessario), quindi esportare, importare o caricare dati, eseguire programmi, report o altri prodotti ORACLE e, quando terminato, rientrare in SQLPLUS.

Anche utenti di SQLPLUS molto esperti a volte devono effettuare delle prove per assicurarsi che linesize, pagesize, sqlprompt e altre caratteristiche e comandi non influiscano negativamente sul codice che viene generato.

Generalmente, l'approccio più corretto è quello di suddividere le operazioni in file di avvio comuni, come off.sql, e in file di avvio dall'elaborazione ben definita, per SQL, SQLPLUS o programmi host. Questi file di avvio devono essere poi tutti controllati da un singolo file di avvio master.

- Capitolo 21
- **Accesso a dati remoti**
-
- 21.1 **Link di database**
- 21.2 **Utilizzo di sinonimi per la trasparenza di locazione**
- 21.3 **Utilizzo della pseudocolonna User nelle viste**
- 21.4 **Collegamenti dinamici: utilizzo del comando copy di SQLPLUS**
- 21.5 **Connessione a un database remoto**
- 21.6 **Strumenti di gestione: Oracle*Names**

A mano a mano che le dimensioni e il numero dei database crescono si presenta la necessità di condividere 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 consentiti dall'utilizzo di *link di database*. Come viene spiegato nel presente capitolo, i link di database consentono all'utente di trattare un gruppo di database distribuiti come un singolo database integrato (l'utilizzo avanzato di link di database per la conservazione di copie remote dei dati è descritto nel Capitolo 28). Nel presente capitolo sono inoltre contenute informazioni sulle connessioni dirette a database remoti, come quelle utilizzate in applicazioni client-server.

21.1 **Link di database**

I *link di database* specificano a ORACLE come passare da un database a un altro. È inoltre possibile definire il percorso di accesso con un metodo appropriato (si consulti il paragrafo “Collegamenti dinamici: utilizzo del comando copy di SQLPLUS” più oltre nel presente capitolo). Se si prevede di utilizzare spesso la stessa connessione a un database remoto, un link di database si rivelerà appropriato.

Funzionamento di un link di database

Un link di database richiede che su ciascuna delle macchine (*host*) coinvolte nell'accesso al database remoto sia avviato SQL*Net.

Quest'ultimo viene normalmente avviato dall'amministratore del database (DBA) o dal gestore del sistema. Nella Figura 21.1 è riportato un esempio di architettura per l'accesso remoto utilizzando un link di database. In tale figura sono visualizzati due host, su ciascuno dei quali viene eseguito SQL*Net. Su ciascuno degli host è presente un database. Per stabilire una connessione tra il primo database (detto "Locale", sull'host "Branch") e il secondo database (detto "Remoto", sull'host "Headquarters"), verrà utilizzato un link di database.

Il link di database visualizzato nella Figura 21.1 è un componente software che si trova nel database "Locale".

I link di database specificano le informazioni di connessione di seguito elencate.

- Il protocollo di comunicazione (ad esempio TCP/IP) da utilizzare durante la connessione.
- L'host su cui risiede il database remoto.
- Il nome del database sull'host remoto.
- Il nome di un account valido nel database remoto.
- La password per tale account.

Al momento dell'utilizzo, un link di database si connetterà effettivamente come utente al database remoto e si disconnetterà ad accesso ai dati remoti completato. Un link di database 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 link di database è descritta più oltre nel presente capitolo, nel paragrafo "Sintassi per link di database".

Utilizzo di un link di database per query remote

L'utente del database "locale" visualizzato nella Figura 21.1 è in grado di accedere a oggetti nel database "remoto" mediante un link di database. Allo scopo è suffi-

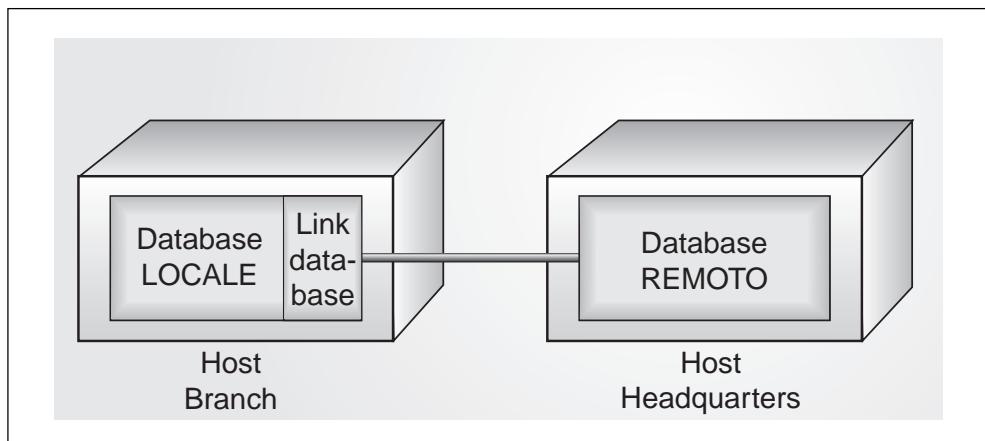


Figura 21.1 Esempio di architettura di un link di database.

ciente aggiungere il nome del link di database al nome di qualsiasi tabella o vista accessibile all'account remoto. Per aggiungere il nome di un link di database al nome di una tabella o di una vista, è necessario far precedere il nome del link di database dal simbolo “@” (pronuncia: “at”) come mostrato nel seguito.

Per le tabelle locali:

```
select *
  from LAVORATORE;
```

Per le tabelle remote, occorre utilizzare un link di database di nome REMOTE_CONNECT:

```
select *
  from LAVORATORE@REMOTE_CONNECT;
```

Quando viene utilizzato il link di database nell'ultima delle precedenti query, ORACLE si connette al database specificato dal link utilizzando il nome utente e la password forniti dal collegamento. ORACLE effettua quindi una query alla tabella LAVORATORE in tale account e restituisce i dati all'utente che aveva originato la query (Figura 21.2).

Il link di database REMOTE_CONNECT nella Figura 21.2 si trova nel database “locale”.

Come illustrato visualizzato in Figura 21.2, la connessione al database “locale” e l'utilizzo del link di database REMOTE_CONNECT nella clausola from restituirà lo stesso risultato di una connessione diretta al database remoto e dell'esecuzione della query senza link di database. Il database remoto appare come locale.

Il massimo numero di link di database che possono essere utilizzati in una singola query è definito mediante il parametro OPEN-LINKS nel file di inizializzazione INIT.ORA del database. Tale parametro è normalmente impostato al numero 4.

Esistono restrizioni alle query eseguite utilizzando link di database. Occorre evitare di utilizzare link di database in query che utilizzano le parole chiave connect by, start with e prior. Alcune query che utilizzano tali parole chiave possono funzionare (ad esempio se prior non viene utilizzata all'esterno della clausola connect by e start

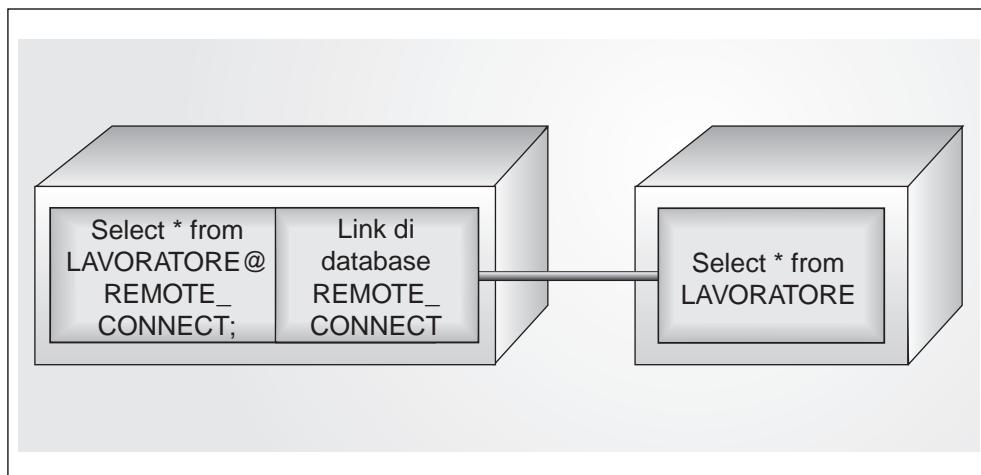


Figura 21.2 Utilizzo di un link di database per una query remota.

with non utilizza una sottoquery), ma la maggior parte delle query strutturate ad albero non funzionerà utilizzando link di database.

Utilizzo di link di database per sinonimi e viste

È possibile creare sinonimi locali e viste che facciano riferimento a oggetti remoti. Allo scopo è necessario creare un riferimento al nome del link di database, preceduto dal simbolo “@”, ogniqualvolta si fa riferimento a una tabella remota. L'esempio seguente mostra come procedere per i sinonimi. Il comando `create synonym` nell'esempio viene eseguito da un account nel database “locale”.

```
create synonym LAVORATORE_SYN
  for LAVORATORE@REMOTE_CONNECT;
```

In questo esempio viene creato il sinonimo `LAVORATORE_SYN` per la tabella `LAVORATORE` a cui si accede mediante il link di database `REMOTE_CONNECT`. Ogni volta che tale 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 link di database è ora definito come parte di un oggetto locale (in questo caso di un sinonimo).

Se l'account remoto a cui accede il link di database non possiede la tabella a cui si fa riferimento, potrà essere utilizzato qualsiasi sinonimo disponibile per l'account remoto (sia privato, sia pubblico).

Se non esistono tali sinonimi per una tabella per la quale è stato garantito l'accesso all'account remoto, è necessario specificare il nome del proprietario della tabella nella query, come nell'esempio seguente.

```
create synonym COMPITOLAVORATORE_SYN
  for Talbot.COMPITOLAVORATORE@REMOTE_CONNECT;
```

In questo esempio, l'account remoto utilizzato dal link di database non possiede la tabella `COMPITOLAVORATORE`, né ha un sinonimo con lo stesso nome. L'account remoto possiede però privilegi sulla tabella `COMPITOLAVORATORE` di proprietà dell'utente remoto `Talbot`. Per questo motivo vengono specificati il proprietario e il nome della tabella: entrambi verranno interpretati nel database remoto. La sintassi per 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 link di database.

Per utilizzare un link di database in una vista è sufficiente aggiungere il collegamento stesso come suffisso ai nomi delle tabelle nel comando `create view`. L'esempio riportato di seguito consente di creare una vista nel database locale di una tabella remota utilizzando il link di database `REMOTE_CONNECT`.

```
create view LOCAL_IMPIEGATO_VIEW
as
select * from LAVORATORE@REMOTE_CONNECT
  where Alloggio = 'ROSE HILL';
```

La clausola from in questo esempio fa riferimento a LAVORATORE@REMOTE_CONNECT. Per questo motivo la tabella base per questa vista non si trova nello stesso database della vista. Si noti inoltre che alla query viene assegnata una clausola where che limita il numero di record restituiti dalla query stessa per la vista.

Questa vista può quindi essere trattata come qualsiasi altra vista del database locale; l'accesso a questa vista può essere consentito ad altri utenti, purché questi abbiano accesso anche al link di database REMOTE_CONNECT.

Utilizzo di un link di database per aggiornamenti remoti

I tipi di aggiornamenti remoti che possono essere effettuati dipendono dalla versione del database ORACLE utilizzato. Se si utilizza ORACLE con l'opzione distribuita, è possibile aggiornare le tabelle in tutti i database ORACLE remoti che eseguano la stessa opzione. Se non si utilizza ORACLE con l'opzione distribuita, l'aggiornamento è limitato ai database remoti che si trovano sullo stesso host del database locale.

La sintassi del link di database per gli aggiornamenti remoti è la stessa utilizzata per le query remote. Occorre aggiungere il nome del link di database al nome della tabella che si intende aggiornare. Ad esempio, per modificare i valori Alloggio per i lavoratori in una tabella remota LAVORATORE, si esegue il comando update riportato di seguito.

```
update LAVORATORE@REMOTE_CONNECT
  set Alloggio = 'CRANMÉR'
  where Alloggio = 'ROSE HILL';
```

Questo comando update utilizzerà il link di database REMOTE_CONNECT per connettersi al database remoto. Il comando aggiornerà quindi la tabella LAVORATORE in tale database, seguendo le condizioni set e where specificate.

È inoltre possibile utilizzare sottoquery nella parte set del comando update (Capitolo 14). La clausola from di tali sottoquery può fare riferimento sia al database locale, sia al database remoto. Per fare riferimento al database remoto in una sottoquery, occorre aggiungere il nome del link di database ai nomi delle tabelle nella clausola from. Di seguito è riportato un esempio.

```
update LAVORATORE@REMOTE_CONNECT      /*in database remoto*/
  set Alloggio =
    (select Alloggio
     from ALLOGGIO@REMOTE_CONNECT /*in database remoto*/
       where Direttore = 'KEN MULLER')
  where Alloggio = 'ROSE HILL';
```

NOTA Se non si aggiunge il nome del link di database ai nomi delle tabelle nella clausola from delle sottoquery update, vengono utilizzate le tabelle nel database locale. Ciò avviene anche se la tabella da aggiornare si trova in un database remoto.

In questo esempio la tabella remota LAVORATORE viene aggiornata basandosi sul valore Alloggio nella tabella remota ALLOGGIO. Se nella sottoquery non

viene utilizzato il link di database, come nell'esempio seguente, viene invece utilizzata la tabella ALLOGGIO del database locale. Se ciò non è desiderato, potrebbe prodursi una sgradita miscela di dati locali nella tabella database remota. Se invece si intende realmente procedere in questo modo, è consigliata un'estrema cautela.

```
update LAVORATORE@REMOTE_CONNECT      /*in database remoto*/
  set Alloggio =
    (select Alloggio
     from ALLOGGIO                  /*in database locale*/
       where Direttore = 'KEN MULLER')
  where Alloggio = 'ROSE HILL';
```

Sintassi per link di database

Un link di database può essere creato con il comando seguente:

```
create [public] database link REMOTE_CONNECT
connect to nomeutente identified by password
using 'stringa_connessione';
```

La sintassi specifica da utilizzare per la creazione di un link di database dipende da due criteri:

- lo stato “pubblico” o “privato” del link di database;
- l'utilizzo di connessioni predefinite o esplicite per il database remoto.

Tali criteri e le sintassi relative verranno descritti nei paragrafi seguenti.

NOTA Per creare un link di database è necessario disporre del privilegio di sistema CREATE DATABASE LINK. L'account a cui ci si connette nel database remoto dovrà disporre del privilegio CREATE SESSION. Ambedue tali privilegi sono compresi nel ruolo CONNECT di ORACLE.

Link di database pubblici e privati Un link di database *pubblico* è disponibile per tutti gli utenti di un database. Al contrario, un link di database *privato* è disponibile solo per l'utente che lo ha creato. Non è possibile per un utente concedere ad altri l'accesso a un link di database privato. Il link di database può essere pubblico (disponibile per tutti gli utenti) o in alternativa privato.

Per definire un link di database come pubblico, si utilizza la parola chiave public nel comando create database link, come nell'esempio riportato di seguito.

```
create public database link REMOTE_CONNECT
connect to nomeutente identified by password
using 'stringa_connessione';
```

NOTA Per creare un link di database pubblico è necessario disporre del privilegio di sistema CREATE PUBLIC DATABASE LINK. Tale privilegio è compreso nel ruolo DBA di ORACLE.

Connessioni predefinite e connessioni esplicite Per la creazione di un link di database non è necessario specificare una clausola “connect to... identified by...”. Se non si specifica tale clausola il link, al momento dell’uso, tenterà di aprire una sessione nel database remoto con lo stesso nome utente e la stessa password dell’account nel database locale. Si parla di *connessione predefinita*, in quanto la combinazione nome utente/password sarà predefinita come la combinazione in uso nel database locale.

Di seguito è riportato un esempio di link di database pubblico creato con connessione predefinita. L’utilizzo delle connessioni predefinite viene descritto più avanti, nel paragrafo “Utilizzo della pseudocolonna User nelle viste”.

```
create public database link REMOTE_CONNECT
using 'stringa_connessione';
```

Se utilizzato, tale link di database tenterà di connettersi al database remoto utilizzando il nome utente e la password correnti. Se il nome utente corrente non è valido per il database remoto, o se la password è differente, il tentativo di connessione fallirà insieme all’istruzione SQL che utilizza il link.

Una connessione *esplicita* specifica un nome utente e una password che il link utilizzerà durante la connessione al database remoto. A prescindere da quale account locale utilizzi il link, verrà utilizzato sempre lo stesso account remoto. Di seguito è riportata la creazione di un link di database con connessione esplicita.

```
create public database link REMOTE_CONNECT
connect to WAREHOUSE identified by ACCESS339
using 'stringa_connessione';
```

Questo esempio mostra un comune utilizzo delle connessioni esplicite nei link di database. Nel database remoto è stato creato un utente di nome Warehouse, a cui è stata assegnata la password ACCESS339. All’account Warehouse potrà quindi essere consentito l’accesso SELECT a specifiche tabelle, esclusivamente per l’utilizzo mediante link di database. Il link di database REMOTE_CONNECT fornirà quindi a tutti gli utenti locali l’accesso all’account Warehouse.

Sintassi della stringa di connessione SQL*Net (sia SQL*Net V2, sia Net8) utilizza *nomi di servizio* per identificare le connessioni remote. I dettagli di connessione per tali nomi di servizio sono contenuti in file distribuiti a ciascun host della rete. Quando incontra un nome di servizio, ORACLE verifica il file SQL*Net locale (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.

Per utilizzare SQL*Net è necessario conoscere il nome del servizio che punta al database remoto. Ad esempio, se il nome di servizio ‘HQ’ specifica i parametri di connessione per il database necessario, la stringa ‘HQ’ dovrà essere utilizzata come stringa di connessione nel comando create database link. L’esempio seguente mostra un link di database privato che utilizza una connessione predefinita e un nome di servizio SQL*Net.

```
create database link REMOTE_CONNECT
using 'HQ';
```

Quando questo link viene utilizzato, ORACLE verifica il file TNSNAMES.ORA sull'host locale per determinare il database a cui connettersi. Per tentare la connessione a tale database vengono utilizzati il nome utente e la password correnti.

I file TNSNAMES.ORA per una rete di database dovrebbero essere coordinati dai DBA. Nell'esempio seguente è riportata una tipica voce del file TNSNAMES.ORA (per una rete che utilizzi il protocollo TCP/IP).

```
HQ =(DESCRIPTION=
      (ADDRESS=
        (PROTOCOL=TCP)
        (HOST=HOST1)
        (PORT=1521))
      (CONNECT DATA=
        (SID=remote)))
```

In questo esempio, il nome di servizio 'HQ' è mappato a un descrittore di connessione. Tale descrittore specifica al database quale protocollo utilizzare (TCP/IP) e a quale host (HOST1) e database (remoto) connettersi. L'informazione "port" fa riferimento alla porta dell'host che verrà utilizzata per la connessione: tale dato è specifico dell'ambiente. Differenti protocolli avranno parole chiave differenti, ma tutti dovranno trasmettere lo stesso contenuto.

21.2 Utilizzo di sinonimi per la trasparenza di locazione

Durante la vita di un'applicazione i suoi dati verranno molto probabilmente spostati da un database a un altro, o da un host a un altro. Per questo motivo la manutenzione risulta semplificata se l'esatta locazione fisica di un oggetto database viene nascosta all'utente (e all'applicazione).

Il miglior modo per integrare tale trasparenza di locazione è rappresentato dall'utilizzo di sinonimi. Invece di scrivere applicazioni (o report SQLPLUS) che contengano query in cui sia specificato il proprietario di una tabella, come:

```
select *
  from Talbot.LAVORATORE;
```

si crea un sinonimo per detta tabella e nella query si fa riferimento a tale sinonimo.

```
create synonym LAVORATORE
  for Talbot.LAVORATORE;
```

```
select *
  from LAVORATORE;
```

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 locazione della tabella verso il database rappresenterà un vantaggio ogni volta che la tabella stessa verrà spostata (ad esempio per il passaggio da un database di sviluppo a un database di verifica).

Oltre a nascondere la proprietà delle tabelle di un'applicazione, è possibile anche nascondere la locazione fisica dei dati, utilizzando link di database e sinonimi. Utilizzando sinonimi locali per tabelle remote, un altro livello di logica viene spostato all'esterno dell'applicazione e all'interno del database. Ad esempio, il sinonimo locale COMPITOLAVORATORE, come di seguito definito, fa riferimento a una tabella collocata in un database differente, su un host differente. Se tale tabella verrà spostata, sarà necessario modificare solo il link, mentre il codice applicativo che utilizza il sinonimo resterà invariato.

```
create synonym COMPITOLAVORATORE
for COMPITOLAVORATORE@REMOTE_CONNECT;
```

Se l'account remoto utilizzato dal link di database 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 COMPITOLAVORATORE
for COMPITOLAVORATORE@REMOTE_CONNECT;
```

dove COMPITOLAVORATORE, nell'account remoto utilizzato dal link di database, è un sinonimo per la tabella COMPITOLAVORATORE di un altro utente.

La seconda possibilità è quella di includere il nome del proprietario remoto nella creazione del sinonimo locale, come nell'esempio seguente.

```
create synonym COMPITOLAVORATORE
for Talbot.COMPITOLAVORATORE@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 manutenzione 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 applicare il comando describe a detta tabella, anche se resterà possibile utilizzare su di essa il comando select. Perché il comando describe funzioni correttamente, sarà necessario utilizzare il formato visualizzato nell'ultimo esempio e specificare il proprietario.

21.3 Utilizzo della pseudocolonna User nelle viste

La pseudocolonna User è molto utile per l'impiego di metodi di accesso a dati remoti. Ad esempio, talvolta si desidera fare in modo che non tutti gli utenti siano in grado di vedere tutti i record di una tabella. Per risolvere tale problema è necessario pensare agli utenti remoti come utenti speciali all'interno del database.

Per imporre le restrizioni sui dati sarà necessario creare una vista a cui accederà l'account remoto.

Ma che cosa è possibile 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 3, 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. La pseudocolonna User, se selezionata, restituisce sempre il nome utente ORACLE che ha eseguito la query. Così, se una colonna della tabella contiene nomi utente, tali valori potranno essere confrontati con la pseudocolonna User per applicare restrizioni ai relativi record, come nell’esempio seguente. In tale esempio la query viene eseguita sulla tabella NOME. Se il valore della relativa colonna Nome coincide con il nome dell’utente che immette la query, vengono restituiti i record.

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

NOTA Per un momento è necessario cambiare punto di vista. Dato che la presente discussione concerne operazioni sul database che possiede la tabella su cui si effettua la query, tale database viene considerato “locale”, mentre agli utenti di altri database sono considerati “remoti”.

Per applicare restrizioni all’accesso remoto a righe della tabella, è necessario in primo luogo considerare quali colonne è opportuno da utilizzare per tali restrizioni. Esistono normalmente divisioni logiche nei dati di una tabella, come Reparto o Prov. Per ciascuna singola divisione, occorre creare un account utente separato nel database locale. Per questo esempio viene aggiunta una colonna Reparto alla tabella LAVORATORE.

```
alter table LAVORATORE
add
(Reparto VARCHAR2(10));
```

Si supponga che nella tabella LAVORATORE siano rappresentati quattro reparti principali e che si crei un account per ciascuno. Sarà quindi possibile impostare ciascun link di database dell’utente remoto per l’utilizzo del proprio specifico account utente nel database locale. Per questo esempio si suppone che i reparti si chiamino NORD, EST, SUD e OVEST. Per ciascuno dei reparti dovrà essere creato uno specifico link di database. Ad esempio, i membri del reparto SUD utilizzeranno il link di database riportato nell’esempio seguente.

```
create database link SUD_LINK
connect to SUD identified by PAW
using 'HQ';
```

Il link di database in questo esempio è privato con connessione esplicita all’account SUD nel database remoto.

Gli utenti remoti che eseguano query mediante i propri link di database (come SUD_LINK nell’esempio precedente), verranno connessi al database HQ, con il nome del proprio reparto (come SUD) per nome utente. Così il valore della colonna User, per qualsiasi tabella su cui l’utente esegua una query, sarà ‘SUD’.

Si crei ora una vista della tabella di base, confrontando la pseudocolonna User con il valore della colonna Reparto nella clausola where della vista.

```
create or replace view RESTRICTED_LAVORATORE
  as select *
    from LAVORATORE
   where Reparto = User;
```

Questo uso della pseudocolonna User è stato descritto per la prima volta nel Capitolo 18.

Un utente che si connetta mediante il link di database SUD_LINK, e venga così connesso come utente SUD, sarà in grado di vedere solo i record di LAVORATORE che abbiano un valore di Reparto uguale a 'SUD'. Se gli utenti accedono alla tabella da un database remoto, le loro connessioni avvengono attraverso link di database e gli account locali da essi utilizzati sono noti all'operatore, che li ha impostati.

Questo tipo di restrizione può anche essere imposta nel database remoto invece che nel database in cui la tabella risiede. Gli utenti nel database remoto possono creare viste entro i propri database utilizzando il formato di seguito descritto.

```
create or replace view SUD_LAVORATORI
  as select *
    from LAVORATORE@REMOTE_CONNECT
   where Reparto = 'SUD';
```

In questo caso le restrizioni Reparto restano in vigore, ma vengono 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.

21.4 Collegamenti dinamici: utilizzo 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. Il comando, anche se consente all'utente di selezionare le colonne da copiare, funziona meglio quando vengano selezionate tutte le colonne di una tabella. In breve, il grande vantaggio nell'utilizzo di questo comando consiste nella 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 (ad esempio LAVORATORE). Come si procede se la tabella LAVORATORE contiene 100000 righe che occupano un totale di 100 MB di spazio e si desidera creare una copia della tabella stessa in un database differente? Il metodo più semplice comporta la creazione di un link di database e l'utilizzo dello stesso in un comando create table... as select. Tale opzione è riportata nell'esempio seguente.

```
create database link REMOTE_CONNECT
```

```

connect to TALBOT identified by REGISTRO
using 'HQ';

create table LAVORATORE
as
select * from LAVORATORE@REMOTE_CONNECT;

```

Il primo comando crea il link di database, mentre il secondo crea una nuova tabella basata su tutti i dati della tabella remota.

Sfortunatamente questa opzione genera una transazione molto grande (tutte le 100000 righe vengono sottoposte al comando `insert` e inserite nella nuova tabella come una singola transazione). Ciò causa un sensibile carico su strutture interne di ORACLE dette *segmenti di rollback*. I segmenti di rollback memorizzano l'immagine precedente dei dati sino a che questi sono trasmessi al database. Dato che questa tabella verrà popolata da un singolo comando `insert`, verrà generata una singola transazione di grandi dimensioni, che con tutta probabilità supererà lo spazio disponibile nei segmenti di rollback correnti. Tale insuccesso causerà a sua volta il fallimento della creazione della tabella.

Per suddividere la transazione in porzioni più piccole, occorre utilizzare il comando SQLPLUS `copy`. La sintassi per tale comando è riportata di seguito.

```

copy from
[remote nomeutente/remoto password@stringa connessione]
[to nomeutente/password@stringa connessione]
{append|create|insert|replace}
nome tabella
using sottoquery;

```

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 la sorgente dei dati copiati, non sono necessarie le informazioni di connessione remota per la sorgente dei dati.

Per impostare le dimensioni delle voci di transazione si utilizza il comando SQLPLUS `set` impostando un valore per il parametro `arraysize`. Ciò consente di definire il numero di record che verranno recuperati in ciascun “batch”. Il parametro `copycommit` specifica a SQLPLUS quanti batch per volta dovranno essere trasmessi. Lo script SQLPLUS riportato di seguito esegue la stessa copia dei dati consentita dal comando `create table as`, ma suddivide la singola transazione in più transazioni. Nell'esempio seguente i dati sono trasmessi dopo ciascun gruppo di 1000 record. Ciò riduce le dimensioni necessarie per i segmenti di rollback della transazione da 100 MB a 1 MB.

```

set copycommit 1
set arrayszie 1000

copy from TALBOT/REGISTRO@HQ -
create LAVORATORE -
using -
select * from LAVORATORE

```

NOTA Eccettuata l'ultima, ciascuna riga 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 21.1.

Il feedback messo a disposizione da questo comando può inizialmente confondere. Completato l'ultimo commit, il database riporta 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).

Tabella 21.1 Opzioni per il 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. |

21.5 Connessione a un database remoto

Oltre alle connessioni tra database descritte in precedenza nel presente capitolo, è anche possibile effettuare connessioni dirette a un database remoto mediante uno strumento di ORACLE. Così, invece di immettere:

```
sqlplus nomeutente/password
```

e accedere al database locale, è possibile accedere direttamente a un database remoto. Allo scopo, occorre immettere il proprio nome utente e la propria password insieme alla stringa di connessione SQL*Net per il database remoto:

```
sqlplus nomeutente/password@HQ
```

Tale comando causa la connessione diretta con il database “HQ”. La configurazione dell’host per questo tipo di connessione è illustrata nella Figura 21.3. L’host “Branch” dispone degli strumenti di ORACLE (come SQLPLUS) e sta eseguendo SQL*Net. L’host “Remote” sta eseguendo SQL*Net e dispone di un database ORACLE. Sull’host “Branch” può essere o meno presente un database. La specificazione della stringa di connessione SQL*Net al database remoto forza ORACLE a ignorare qualsiasi database locale.

Come si vede nella Figura 21.3, i requisiti hardware per l’host “Branch” sono molto ridotti. Tale host deve supportare solo lo strumento front-end e SQL*Net: una tipica configurazione per applicazioni client-server. La macchina client, come

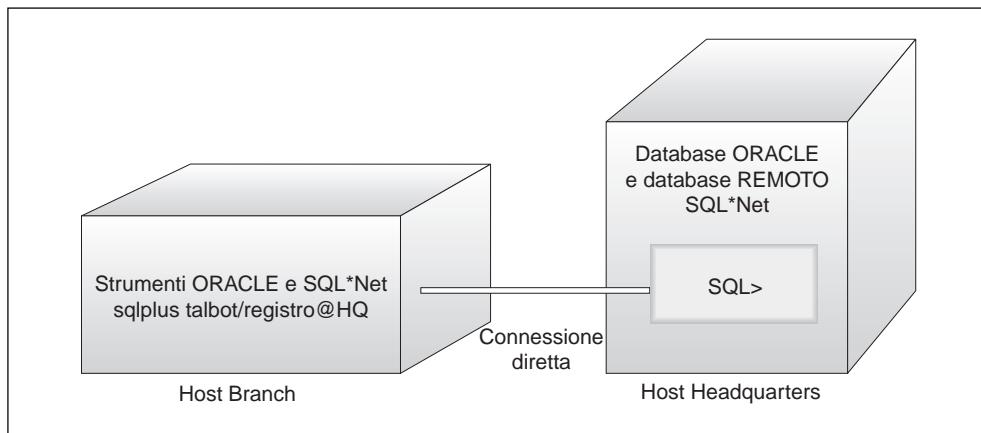


Figura 21.3 Esempio di architettura per una connessione remota.

l'host “Branch”, viene utilizzata principalmente per la presentazione dei dati mediante gli strumenti di accesso al database. Il lato del server, come avviene per l'host “Headquarters” viene utilizzato per la manutenzione dei dati e l'elaborazione delle richieste di accesso agli stessi da parte degli utenti.

21.6 Strumenti di gestione: Oracle*Names

Nella manutenzione degli accessi ai dati remoti sono coinvolte numerose operazioni di gestione. Nel presente capitolo sono stati descritti i sinonimi, i link di database, le viste e i nomi utente guidati dai dati. All'interno dei link di database è necessario gestire nomi di servizio SQL*Net e file TNSNAMES.ORA. Lo strumento Oracle*Names, rilasciato insieme a SQL*Net 2.1, può essere utilizzato per numerosi di tali operazioni di gestione.

Con Oracle*Names tutte le operazioni di gestione dei database distribuiti sono gestite mediante un servizio globale di assegnazione dei nomi, disponibile a tutti gli host di una rete. Questo viene utilizzato per salvare informazioni su:

- descrittori di connessione;
- link di database;
- alias di oggetti.

Ciò cambia il modo in cui vengono risolti i link di database. In precedenza, quando veniva specificato un link di database, il database controllava per primi i link privati dell'utente. Se tra essi non veniva trovato alcun link il cui nome coincidesse con quello cercato, venivano controllati i link di database pubblici disponibili.

Oracle*Names aggiunge un ulteriore livello a tale procedimento. Ora, se i primi due controlli non restituiscano un nome di link di database che corrisponda, il link viene ricercato nell'elenco di nomi di link globali del server Oracle*Names.

Se il nome del link viene trovato lì, Oracle*Names restituisce le specifiche del link e risolve la query.

Ciò semplifica notevolmente l'amministrazione della *trasparenza di locazione*: l'abilità di mascherare la collocazione fisica dei dati in un ambiente distribuito. Le informazioni relative all'accesso a dati remoti vengono ora memorizzate in una collocazione centrale. L'effetto si manifesta ogniqualvolta viene modificata una parte della collocazione di un oggetto. Se ad esempio più collegamenti utilizzassero connessioni esplicite a un singolo database remoto, una modifica alla password dell'utente avrebbe richiesto, prima che esistesse Oracle*Names, l'eliminazione e la nuova creazione di più link di database. Con Oracle*Names, tale modifica avviene una sola volta.

Oracle*Names supporta inoltre la struttura DNS (Domain Name Service) presentata da Oracle con SQL*Net V2. La struttura DNS consente l'organizzazione gerarchica degli host di una rete. Ciascun nodo all'interno dell'organizzazione viene detto *dominio* e ciascun dominio riceve un'etichetta a seconda della propria funzione; tra queste vi sono ad esempio "COM" per le società commerciali ed "EDU" per le scuole. Ciascun dominio può prevedere sottodomini. A ciascun server viene assegnato un nome unico all'interno della rete: tale nome contiene informazioni sul modo in cui il server si inserisce nella gerarchia della rete stessa. La struttura DNS di Oracle consente la specificazione di gerarchie di rete. Così un server potrà essere identificato come HR.HQ.ACME.COM, ovvero come il server HR della rete HQ della società ACME.

Se i descrittori di connessione sono anch'essi memorizzati in Oracle*Names, la necessità di manutenzione manuale su più copie del file TNSNAMES.ORA diminuisce rapidamente. Il server centralizzato Oracle*Names definisce le relazioni tra gli oggetti della rete. La rete di database può essere suddivisa in regioni amministrative e le operazioni di gestione possono essere suddivise di conseguenza. Una modifica a una regione verrà propagata in modo trasparente a tutte le altre.

Per l'integrazione di un'applicazione che utilizzi SQL*Net, Oracle*Names può significare una semplificazione dell'amministrazione. Si consiglia di verificare insieme agli amministratori di sistema e ai DBA se Oracle*Names è disponibile.

•
•
•
•
• Capitolo 22

• **Introduzione a PL/SQL**

- •
• 22.1 **PL/SQL: nozioni di base**
• 22.2 **La sezione delle dichiarazioni**
• 22.3 **La sezione dei comandi eseguibili**
• 22.4 **La sezione per la gestione delle eccezioni**
•

PL/SQL è un linguaggio strutturato di query (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 proprie regole aziendali mediante la creazione di procedure memorizzate 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, procedure memorizzate e package sono descritti nei capitoli seguenti di questa parte del libro. Nel presente capitolo sono descritte le più semplici strutture e sintassi utilizzate nel linguaggio PL/SQL.

22.1 **PL/SQL: nozioni di base**

Il codice PL/SQL è raggruppato in strutture dette *blocchi*. Se si crea una procedura memorizzata 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, il blocco stesso viene detto *anonimo*. Gli esempi riportati nel presente capitolo contengono blocchi anonimi di codice PL/SQL. I capitoli seguenti descriveranno la creazione di blocchi dotati di nome.

Un blocco di codice PL/SQL contiene tre sezioni, descritte nella Tabella 22.1.

Tabella 22.1 Sezioni di un blocco anonimo PL/SQL.

| SEZIONE | DESCRIZIONE |
|--------------------------|--|
| Dichiarazioni | Definisce e inizializza le variabili e i cursori utilizzati nel blocco. |
| Comandi eseguibili | Utilizza comandi di controllo del flusso (come if e loop) per eseguire i comandi e assegnare valori alle variabili dichiarate. |
| Gestione delle eccezioni | Mette a disposizione una gestione personalizzata delle condizioni d'errore. |

All'interno di un blocco PL/SQL la prima sezione è quella delle dichiarazioni, dove vengono definite le variabili e i cursori che verranno 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 della 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 successivi paragrafi del presente capitolo viene descritta ciascuna sezione del blocco PL/SQL.

22.2 La sezione delle dichiarazioni

La sezione delle dichiarazioni dà inizio a un blocco PL/SQL. Tale 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 esistenti e risultati di query, come negli esempi seguenti.

Nell'esempio riportato di seguito viene calcolata l'area di un cerchio. Il risultato viene memorizzato in una tabella di nome `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) := 3.1415926;
  radius INTEGER(5);
  area   NUMBER(14,2);
begin
  radius := 3;
  area := pi*power(radius,2);
  insert into AREA values (radius, area);
end;
.
```

Il simbolo ‘.’ indica la fine del blocco PL/SQL e il simbolo ‘/’ causa l'esecuzione del blocco PL/SQL stesso. Quando il blocco PL/SQL viene eseguito, ORACLE genera la risposta riportata di seguito.

PL/SQL procedure successfully completed.

Per verificare che il blocco PL/SQL sia stato completato correttamente, è possibile selezionare nel database le righe che sono state inserite mediante il codice PL/SQL. La query e i risultati nell'esempio seguente riportano la riga creata dal blocco PL/SQL nella tabella AREA.

```
select *
  from AREA;

  RADIUS      AREA
----- -----
      3        28.27
```

L'output mostra che il blocco PL/SQL ha inserito nella tabella AREA una singola riga. È stata creata una sola riga in quanto era stato specificato un solo valore per il raggio.

Nella prima sezione del blocco PL/SQL, riportata nell'esempio seguente, vengono dichiarate tre variabili. È necessario dichiarare le variabili che verranno utilizzate nella sezione dei comandi eseguibili del blocco PL/SQL.

```
declare
  pi      constant NUMBER(9,7) := 3.1415926;
  radius  INTEGER(5);
  area    NUMBER(14,2);
```

La prima variabile dichiarata è *pi*, impostata su un valore costante mediante la parola chiave *constant*. Il valore viene assegnato per mezzo dell'operatore *:=*.

```
pi      constant NUMBER(9,7) := 3.1415926;
```

Le successive due variabili sono definite, ma a esse non vengono assegnati valori predefiniti.

```
radius  INTEGER(5);
area    NUMBER(14,2);
```

Nella sezione delle dichiarazioni è possibile assegnare un valore iniziale a una variabile. Occorre far seguire la relativa specificazione del tipo di dati dall'assegnazione del valore, come nell'esempio seguente.

```
radius 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.

Nell'esempio seguente viene dichiarato un cursore per prelevare un record dalla tabella RADIUS_VALS. Il cursore viene dichiarato nella sezione delle dichiarazioni. Viene inoltre dichiarata una variabile di nome "rad_val" e tipo di dati basato sui risultati del cursore.

```
declare
  pi      constant NUMBER(9,7) := 3.1415926;
  area   NUMBER(14,2);
  cursor rad_cursor is
    select * from RADIUS_VALS;
```

```

rad_val rad_cursor%ROWTYPE;
begin
  open rad_cursor;
  fetch rad_cursor into rad_val;
  area := pi*power(rad_val.radius,2);
  insert into AREA values (rad_val.radius, area);
  close rad_cursor;
end;
/

```

In questo esempio la tabella RADIUS_VALS contiene una singola colonna (di nome Radius) e una singola riga (con un valore di Radius pari a 3).

Nella prima parte della sezione delle dichiarazioni vengono definite le variabili *pi* e *Area*, come nei precedenti esempi del presente capitolo. La variabile “radius” non è definita; viene invece definito un cursore denominato “rad_cursor”. La definizione del cursore consiste in un nome per il cursore stesso (“rad_cursor”) e in una query (select * from RADIUS_VALS;). Un cursore conserva i risultati di una query perché questi siano elaborati da altri comandi all'interno del blocco PL/SQL:

```

declare
  pi      constant NUMBER(9,7) := 3.1415926;
  area   NUMBER(14,2);
  cursor rad_cursor is
    select * from RADIUS_VALS;

```

Una dichiarazione finale crea una variabile la cui struttura viene ereditata dalla serie dei risultati del cursore.

```
rad_val rad_cursor%ROWTYPE;
```

La variabile “rad_val” sarà in grado di fare riferimento a ciascuna colonna della serie di risultati della query. In questo esempio la query restituisce solo una singola colonna, ma se la tabella contenesse più colonne sarebbe possibile fare riferimento a tutte queste mediante la variabile “rad_val”.

Oltre alla dichiarazione %ROWTYPE è possibile utilizzare la dichiarazione %TYPE per ereditare informazioni sul 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 nella serie di risultati del cursore. Se si utilizza la dichiarazione %TYPE, la variabile eredita solo la definizione della colonna utilizzata per definire la variabile stessa. È inoltre possibile basare definizioni %TYPE su cursori, come nell'esempio seguente.

```

cursor rad_cursor is
  select * from RADIUS_VALS;
  rad_val rad_cursor%ROWTYPE;
  rad_val_radius  rad_val.radius%TYPE;

```

Nel precedente esempio la variabile “rad_val” eredita i tipi di dati della serie di risultati del cursore “rad_cursor”. La variabile “rad_val_radius” eredita il tipo di dati della colonna Radius all'interno della variabile “rad_val”.

Il vantaggio dell'ereditarietà dei tipi di dati con le definizioni %ROWTYPE e %TYPE consiste nel fatto che ciò rende le definizioni del tipo di dati nel codice PL/SQL indipendenti dalla struttura dei dati sottostante. Se la colonna Radius RADIUS_VALS 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.

22.3 La sezione dei comandi eseguibili

Nella sezione dei comandi eseguibili vengono elaborate le variabili e i cursori dichiarati nella sezione delle dichiarazioni del blocco PL/SQL. Questa sezione inizia sempre con la parola chiave begin. Nell'esempio 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 AREA.

```
declare
    pi      constant NUMBER(9,7) := 3.1415926;
    radius INTEGER(5);
    area   NUMBER(14,2);
begin
    radius := 3;
    area := pi*power(radius,2);
    insert into AREA values (radius, area);
end;
.
/
```

Nell'esempio precedente la sezione dei comandi eseguibili è:

```
begin
    radius := 3;
    area := pi*power(radius,2);
    insert into AREA values (radius, area);
end;
```

Dopo la parola chiave begin inizia il lavoro del blocco PL/SQL. Per prima cosa viene assegnato un valore alla variabile “radius”. La variabile “radius” e la costante *pi* vengono utilizzate per determinare il valore della variabile “area”. I valori “radius” e “area” vengono quindi immessi nella tabella AREA.

La sezione dei comandi eseguibili del blocco PL/SQL può contenere comandi che eseguono i cursori dichiarati nella sezione delle dichiarazioni. Nell'esempio seguente (dalla sezione delle dichiarazioni del presente capitolo), la sezione dei comandi eseguibili presenta numerosi comandi riguardanti il cursore “rad_cursor”.

```
declare
    pi      constant NUMBER(9,7) := 3.1415926;
    area   NUMBER(14,2);
    cursor rad_cursor is
        select * from RADIUS_VALS;
```

```

rad_val rad_cursor%ROWTYPE;
begin
  open rad_cursor;
  fetch rad_cursor into rad_val;
  area := pi*power(rad_val.radius,2);
  insert into AREA values (rad_val.radius, area);
  close rad_cursor;
end;
/

```

Nel primo comando riguardante il cursore viene utilizzato `open`:

```
open rad_cursor;
```

Quando il cursore “`rad_cursor`” viene sottoposto al comando `open`, la query dichiarata per tale cursore viene eseguita e vengono identificati i record da restituire. Quindi i record vengono estratti dal cursore con il comando `fetch`.

```
fetch rad_cursor into rad_val;
```

Nella sezione delle dichiarazioni la variabile “`rad_val`” era stata dichiarata in modo da ereditare i propri tipi di dati dal cursore “`rad_cursor`”.

```

cursor rad_cursor is
  select * from RADIUS_VALS;
  rad_val rad_cursor%ROWTYPE;
```

Se si estrae con `fetch` un record dal cursore per immetterlo nella variabile “`rad_val`”, 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 stesso con il comando `close`, come nell’esempio seguente.

```
close rad_cursor;
```

La sezione dei comandi eseguibili può contenere logica condizionale come comandi `if` e cicli. Nei paragrafi successivi verranno riportati esempi di ciascuno dei principali tipi di operazioni di controllo di flusso consentiti nel linguaggio PL/SQL. Le operazioni di controllo del flusso possono essere utilizzate per modificare il modo in cui sono gestiti i record sottoposti a `fetch` e i comandi eseguibili.

Logica condizionale

All’interno del linguaggio PL/SQL è possibile utilizzare comandi `if`, `else` ed `elsif` per controllare il flusso di esecuzione all’interno della sezione dei comandi eseguibili. Le sintassi dei comandi di logica condizionale disponibili (esclusi i cicli, descritti nel prossimo paragrafo) sono elencate di seguito.

```

if  <condizione>
  then <comando>
elsif <condizione>
```

```

    then <comando>
else <comando>
end if;
```

Le condizioni if possono essere annidate le une entro le altre, come nell'esempio seguente:

```

if <condizione>
then
  if <condizione>
  then <comando>
  end if;
else <comando>
end if;
```

Annidando le condizioni if è possibile sviluppare rapidamente complessi flussi logici all'interno della sezione dei comandi eseguibili. Occorre comunque 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 utilizzato nel paragrafo precedente, relativo all'area del cerchio, viene ora modificato in modo da contenere logica condizionale. Nell'esempio seguente, la sezione dei comandi eseguibili del blocco PL/SQL è stata modificata in modo da contenere comandi if e then.

```

declare
  pi    constant NUMBER(9,7) := 3.1415926;
  area  NUMBER(14,2);
  cursor rad_cursor is
    select * from RADIUS_VALS;
  rad_val rad_cursor%ROWTYPE;
begin
  open rad_cursor;
  fetch rad_cursor into rad_val;
  area := pi*power(rad_val.radius,2);
  if area >30
  then
    insert into AREA values (rad_val.radius, area);
  end if;
  close rad_cursor;
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 elencati di seguito.

```

if area >30
then
  insert into AREA values (rad_val.radius, area);
end if;
```

I comandi di controllo del flusso in questo esempio iniziano una volta determinato il valore della variabile “area”. Se il valore è maggiore di 30 verrà inserito un record nella tabella AREA. Se il valore è minore di 30 non verrà inserito alcun record. Questo tipo di controllo del flusso può essere utilizzato per stabilire quale tra più istruzioni SQL deve essere eseguita, basandosi sulle condizioni specificate nelle condizioni if. Nell'esempio seguente è riportata la sintassi per i controlli di flusso che comprendono comandi SQL.

```
if area>30
  then <comando>
elsif area<10
  then <comando>
else <comando>
end if;
```

Cicli

I loop possono essere utilizzati per elaborare più record all'interno di un singolo blocco PL/SQL. Il linguaggio PL/SQL supporta tre tipi di cicli.

| | |
|----------------|--|
| Ciclo semplice | Un ciclo che si ripete sino a che al suo interno non viene raggiunta un'istruzione exit o exit when. |
| Ciclo FOR | Un ciclo che si ripete per un numero specificato di volte. |
| Ciclo WHILE | Un ciclo che si ripete sino a che non viene soddisfatta una condizione. |

Nel seguito sono riportati esempi per ciascun tipo di cicli. Tali esempi utilizzeranno come punto di partenza i blocchi PL/SQL riportati in precedenza nel presente capitolo. I cicli possono essere utilizzati per elaborare più record di un cursore.

Ciclo semplice Nell'esempio seguente un ciclo semplice viene utilizzato per generare più righe nella tabella AREA. Il ciclo viene avviato dalla parola chiave loop. Per determinare quando uscire dal ciclo viene utilizzata la clausola exit when. Una clausola end loop segnala la fine del ciclo.

```
declare
  pi    constant NUMBER(9,7) := 3.1415926;
  radius INTEGER(5);
  area   NUMBER(14,2);
begin
  radius := 3;
  loop
    area := pi*power(radius,2);
    insert into AREA values (radius, area);
    radius := radius+1;
    exit when area >100;
  end loop;
end;
/
```

La sezione loop dell'esempio stabilisce il controllo di 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.

```

loop
    /* Calcola l'area in base al valore del raggio.      */
    area := pi*power(radius,2);
    /* Inserisce i valori correnti nella tabella AREA.   */
    insert into AREA values (radius, area);
    /* Incrementa il valore del raggio di 1.            */
    radius := radius+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 AREA. Il primo record sarà quello generato da un valore di Radius pari a 3. Quando il valore Area supera 100, non vengono inseriti altri record nella tabella AREA.

Di seguito è riportato un esempio di output successivo all'esecuzione del blocco PL/SQL.

```

select *
  from AREA
 order by Radius;

```

| RADIUS | AREA |
|--------|-------|
| 3 | 28.27 |
| 4 | 50.27 |
| 5 | 78.54 |
| 6 | 113.1 |

Dato che il valore di Area per un valore di Radius pari a 6 supera 100, non vengono elaborati ulteriori valori di Radius e il blocco PL/SQL viene completato.

Ciclo semplice a cursore Gli attributi di un cursore, come l'esistenza o meno di righe rimaste da trasmettere con fetch, possono essere utilizzati come criteri di uscita da un ciclo. Nell'esempio seguente, un cursore viene eseguito fino a quando la query non restituisce più altre righe. Per determinare lo stato del cursore vengono verificati i relativi attributi. I cursori sono dotati di quattro attributi, che possono essere utilizzati per la programmazione.

- %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 booleani e vengono impostati su VERO o FALSO. Poiché detti attributi sono booleani, è possibile valutare le loro impostazioni senza farli esplicitamente coincidere con valori di VERO o FALSO. Ad esempio, il comando di seguito riportato causa un output quando rad_cursor%NOTFOUND è VERO:

```
exit when rad_cursor%NOTFOUND;
```

Nella stringa seguente viene utilizzato un ciclo semplice per elaborare più righe di un cursore.

```
declare
    pi      constant NUMBER(9,7) := 3.1415926;
    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 AREA values (rad_val.radius, area);
    end loop;
    close rad_cursor;
end;
.
```

La sezione di ciclo del blocco PL/SQL esegue la stessa elaborazione del ciclo semplice descritto nel precedente paragrafo, con un'eccezione. Invece di basare il criterio 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 VERO e ciò causerà l'uscita dal ciclo. Di seguito è riportata la versione commentata del ciclo stesso.

```
loop
    /* Nel ciclo, preleva un record.          */
    fetch rad_cursor into rad_val;
    /* Se il tentativo di prelevamento non trova */
    /* record nel cursore, esce dal ciclo.       */
    exit when rad_cursor%NOTFOUND;
    /* Se il tentativo restituisce un record,    */
    /* elabora il valore del raggio e inserisce */
    /* un record nella tabella AREA.            */
    area := pi*power(rad_val.radius,2);
    insert into AREA values (rad_val.radius, area);
    /* Segnala la fine del ciclo.                */
end loop;
```

Quando il precedente blocco PL/SQL viene eseguito, tutti i record della tabella RADIUS_VALS vengono elaborati dal ciclo. Finora la tabella RADIUS_VALS conteneva solo un record: un valore di Radius pari a 3. Prima di eseguire il blocco PL/SQL per questa sezione, si aggiungano due nuovi valori di Radius alla tabella RADIUS_VALS: 4 e 10.

Di seguito viene illustrata l'aggiunta dei nuovi record alla tabella RADIUS_VALS.

```
insert into RADIUS_VALS values (4);
insert into RADIUS_VALS values (10);
commit;
```

```
select *
  from RADIUS_VALS
order by Radius;
```

| RADIUS |
|--------|
| 3 |
| 4 |
| 10 |

Una volta aggiunti i nuovi record alla tabella RADIUS_VALS, si esegua il blocco PL/SQL descritto in precedenza nel presente paragrafo. Di seguito è riportato l'output.

```
select *
  from AREA
order by Radius;
```

| RADIUS | AREA |
|--------|--------|
| 3 | 28.27 |
| 4 | 50.27 |
| 10 | 314.16 |

La query della tabella AREA mostra come ogni record della tabella RADIUS_VALS sia stato trasmesso dal cursore ed elaborato. Una volta esauriti i record da elaborare, si è usciti dal ciclo e il blocco PL/SQL è stato completato.

Ciclo FOR Nei cicli semplici il corpo dei comandi viene eseguito finché non viene soddisfatta una condizione di uscita. In un ciclo FOR, il ciclo viene eseguito per un numero di volte specificato. Di seguito è riportato un esempio di ciclo FOR. L'inizio del ciclo FOR viene indicato dalla parola chiave for, seguita dai criteri utilizzati per determinare quando uscire. Dato che il numero di esecuzioni del ciclo viene impostato all'inizio del ciclo stesso, non è necessario un comando exit all'interno del ciclo.

Nell'esempio seguente le aree dei cerchi vengono calcolate basandosi su valori di Radius da 1 a 7 compresi.

```
declare
  pi      constant NUMBER(9,7) := 3.1415926;
```

```

radius INTEGER(5);
area NUMBER(14,2);
begin
  for radius in 1..7 loop
    area := pi*power(radius,2);
    insert into AREA values (radius, area);
  end loop;
end;
.
/

```

I passaggi che l'elaborazione del ciclo comporta sono visualizzati nel seguente esempio commentato.

```

/* Specifica i criteri per il numero di esecuzioni */
/* del ciclo. */
for radius in 1..7 loop
  /* Calcola l'area utilizzando il valore corrente */
  /* di Radius. */
  area := pi*power(radius,2);
  /* Inserisce i valori di area e raggio nella tabella */
  /* AREA. */
  insert into AREA values (radius, area);
  /* Segnala la fine del ciclo. */
end loop;

```

Si noti che nel ciclo FOR non esiste una riga:

```
radius := radius+1;
```

Dato che le specifiche del loop determinano:

```
for radius in 1..7 loop
```

i valori di Radius sono già specificati. Per ciascun valore verranno eseguiti tutti i comandi all'interno del ciclo (tali comandi possono comprendere altra logica condizionale, come condizioni if).

Una volta che il ciclo ha completato l'elaborazione di un valore di Radius, vengono verificati i limiti nella clausola for e si passa, a seconda dei casi, a elaborare il successivo valore di Radius o a completare l'esecuzione del ciclo.

Di seguito è riportato un esempio di output ricavato dall'esecuzione del ciclo FOR.

```

select *
  from AREA
 order by Radius;

```

| RADIUS | 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 istruzioni sono eseguite per un numero di volte specificato. In un ciclo FOR a cursore, i risultati di una query vengono utilizzati per determinare dinamicamente il numero di esecuzioni del ciclo. In questi cicli l'apertura, la trasmissione e la chiusura dei cursori sono eseguite implicitamente. Non è pertanto necessario utilizzare comandi esplicativi per tali azioni.

Di seguito è riportato un ciclo FOR a cursore che effettua una query sulla tabella RADIUS_VALS e inserisce record nella tabella AREA:

```

declare
    pi      constant NUMBER(9,7) := 3.1415926;
    area   NUMBER(14,2);
    cursor rad_cursor is
        select * from RADIUS_VALS;
    rad_val rad_cursor%ROWTYPE;
begin
    for rad_val in rad_cursor
    loop
        area := pi*power(rad_val.radius,2);
        insert into AREA values (rad_val.radius, area);
    end loop;
end;
/

```

In un ciclo FOR a cursore non compare un comando open o fetch. Il comando:

```
for rad_val in rad_cursor
```

apre implicitamente il cursore “rad_cursor” e trasmette un valore nella variabile “rad_val”.

Quando nel cursore non ci sono più record, l'elaborazione esce dal ciclo e il cursore viene chiuso. In un ciclo FOR a cursore non è necessario il comando close.

Di seguito è riportata la parte relativa al ciclo del blocco PL/SQL, con commenti che descrivono il flusso dei controlli. Il ciclo viene controllato in base all'esistenza di un record trasmissibile nel cursore “rad_cursor”. Non è necessario verificare l'attributo %NOTFOUND del cursore; provvede automaticamente il ciclo FOR a cursore.

```

/* Se un record può essere prelevato dal cursore, */
/* lo memorizza nella variabile rad_val. Se          */
/* non ci sono righe da prelevare, salta il ciclo. */
for rad_val in rad_cursor
/* Iniziano i comandi di ciclo.                      */
loop
/* Calcola l'area in base al valore del raggio      */
/* e inserisce un record nella tabella AREA.          */
area := pi*power(rad_val.radius,2);
insert into AREA values (rad_val.radius, area);

```

```
/* Segnala la fine dei comandi di ciclo. */  
end loop;
```

Di seguito è riportato un esempio di output. In questo caso la tabella RADIUS_VALS dispone di tre record, con valori di Radius pari a 3, 4 e 10.

```
select *  
from RADIUS_VALS  
order by Radius;
```

| RADIUS |
|--------|
| 3 |
| 4 |
| 10 |

L'esecuzione del blocco PL/SQL con il ciclo FOR a cursore genera i seguenti record nella tabella AREA:

```
select *  
from AREA  
order by Radius;
```

| RADIUS | AREA |
|--------|--------|
| 3 | 28.27 |
| 4 | 50.27 |
| 10 | 314.16 |

Ciclo WHILE In un ciclo WHILE i comandi sono ripetuti finché non viene soddisfatta una condizione di uscita. Invece di specificare la condizione di uscita per mezzo di un comando exit all'interno del ciclo, la si specifica nel comando while con cui inizia il ciclo.

Nell'esempio seguente viene creato un ciclo WHILE che consenta l'elaborazione di più valori di Radius. Se il valore corrente della variabile Radius soddisfa la condizione while nella specificazione del ciclo, i comandi del ciclo stesso vengono elaborati. Quando un valore di Radius non soddisfa la condizione while nella specificazione del ciclo, l'esecuzione di quest'ultimo termina.

```
declare  
    pi      constant NUMBER(9,7) := 3.1415926;  
    radius INTEGER(5);  
    area    NUMBER(14,2);  
begin  
    radius := 3;  
    while radius<=7  
    loop  
        area := pi*power(radius,2);  
        insert into AREA values (radius, area);  
        radius := radius+1;  
    end loop;  
end;  
/
```

Il ciclo WHILE è simile nella struttura al 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 Radius.      */
radius := 3;
/* Stabilisce i criteri per l'uscita dal ciclo          */
/* Se la condizione è verificata, esegue i comandi      */
/* all'interno del ciclo, altrimenti lo termina.        */
while radius<=7
    /* Inizio dei comandi da eseguire.                  */
loop
    /* Calcola l'area in base al valore corrente di      */
    /* Radius e inserisce un record nella tabella AREA   */
    area := pi*power(radius,2);
    insert into AREA values (radius, area);
    /* Imposta un nuovo valore per la variabile Radius. Questo */
    /* nuovo valore viene confrontato con i criteri di uscita */
    /* e si ha una nuova esecuzione oppure l'uscita dal ciclo */
    radius := radius+1;
    /* Segnala la fine dei comandi del ciclo.            */
end loop;
```

Al momento dell'esecuzione il blocco PL/SQL dell'esempio precedente inserisce record nella tabella AREA. Di seguito è riportato l'output del blocco PL/SQL.

```
select *
  from AREA
 order by Radius;
```

| RADIUS | AREA |
|--------|--------|
| 3 | 28.27 |
| 4 | 50.27 |
| 5 | 78.54 |
| 6 | 113.1 |
| 7 | 153.94 |

A causa del valore assegnato alla variabile “Radius” prima del ciclo, questo è forzato a essere eseguito almeno una volta. È necessario verificare che l'assegnazione della variabile soddisfi le condizioni utilizzate per limitare le esecuzioni del ciclo.

Istruzioni goto

Un'istruzione goto forza il flusso dei comandi a essere immediatamente deviato verso un'altra serie di comandi. Per utilizzare l'istruzione goto è necessario creare in precedenza delle etichette per serie di comandi. Un'istruzione goto non può trasferire il controllo a un blocco annidato all'interno del blocco corrente, all'interno di un ciclo FOR o all'interno di una condizione if.

Per creare un'etichetta occorre racchiuderne il nome tra apici angolari, come nell'esempio seguente.

```
<<area_del_cerchio>>
radius := 3;
while radius<=7
loop
    area := pi*power(radius,2);
    insert into AREA values (radius, area);
    radius := radius+1;
end loop;
```

Nell'esempio precedente l'etichetta `<<area_del_cerchio>>` denomina la serie di comandi che la seguono.

Se all'interno della sezione dei comandi eseguibili compare:

```
goto area_del_cerchio;
```

inizializza l'esecuzione dei comandi all'interno dell'etichetta `<<area_del_cerchio>>`.

Se possibile, è meglio tentare di utilizzare cicli o condizioni `if` al posto delle istruzioni `goto`. I cicli e le condizioni `if`, per loro natura, documentano le regole logiche applicate al flusso del controllo tra i comandi. Un comando `goto`, d'altro canto, non richiede alcuna documentazione circa il motivo per il passaggio a un'altra serie di comandi. Dato che i comandi `goto` non si documentano da sé, è più semplice la manutenzione di codice PL/SQL scritto con condizioni `if` e cicli.

22.4 La sezione per la gestione delle eccezioni

Quando si incontrano eccezioni definite dall'utente o relative al sistema (errori), il controllo del blocco PL/SQL passa alla sezione per la gestione delle eccezioni. All'interno di questa viene utilizzata una clausola `when` 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 per la 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 personalizzate sono riportati nel Capitolo 23 e nel Capitolo 24.

La sezione per la 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. Nell'esempio seguente è riportata la posizione della sezione per la gestione delle eccezioni all'interno del blocco PL/SQL.

```
declare
    <sezione delle dichiarazioni>
begin
    <comandi eseguibili>
exception
    <gestione delle eccezioni>
end;
```

La sezione per la gestione delle eccezioni di un blocco PL/SQL è facoltativa. Nessuno dei blocchi PL/SQL descritti in precedenza nel presente capitolo comprendeva una sezione di questo tipo. In effetti, però, gli esempi precedenti sono basati su una serie molto piccola di valori di input noti, con elaborazione molto ridotta.

Nell'esempio seguente è riportato il ciclo semplice per il calcolo dell'area di un cerchio, con due modifiche (in grassetto). Nella sezione delle dichiarazioni viene dichiarata una nuova variabile di nome “una_variabile” e nella sezione dei comandi eseguibili viene creato un calcolo per determinare il valore della variabile stessa.

```

declare
    pi      constant NUMBER(9,7) := 3.1415926;
    radius INTEGER(5);
    area   NUMBER(14,2);
    una_variabile  NUMBER(14,2);
begin
    radius := 3;
    loop
        una_variabile := 1/(radius-4);
        area := pi*power(radius,2);
        insert into AREA values (radius, area);
        radius := radius+1;
        exit when area >100;
    end loop;
end;
.
/

```

Dato che il calcolo di “una_variabile” comporta la divisione, è possibile incontrare una situazione in cui il calcolo tenti di dividere per zero, con un errore. La prima volta che il ciclo viene eseguito, viene elaborata la variabile Radius (con un valore iniziale pari a 3) e viene inserito un record nella tabella AREA.

Alla seconda esecuzione del ciclo la variabile “Radius” ha valore pari a 4 e il calcolo di “una_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 AREA viene sottoposta a rollback e il blocco PL/SQL termina.

L'elaborazione della condizione di errore può essere modificata aggiungendo una sezione per la gestione delle eccezioni al blocco PL/SQL, come nell'esempio seguente.

```

declare
    pi      constant NUMBER(9,7) := 3.1415926;
    radius INTEGER(5);
    area   NUMBER(14,2);
    some_variable  NUMBER(14,2);
begin

```

```

radius := 3;
loop
  una_variabile := 1/(radius-4);
  area := pi*power(radius,2);
  insert into AREA values (radius, area);
  radius := radius+1;
  exit when area >100;
end loop;
exception
  when ZERO_DIVIDE
    then insert into AREA values (0,0);
end;
.
/

```

Ecco la sezione per la gestione delle eccezioni del blocco PL/SQL:

```

exception
  when ZERO_DIVIDE
    then insert into AREAS values (0,0);

```

Quando si verifica un errore, il blocco PL/SQL ricerca l'eccezione definita nella sezione per la 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 eccezioni definibili dall'utente è possibile utilizzare la clausola `when others` per indirizzare tutte le eccezioni non definite all'interno della sezione per la gestione delle eccezioni. Viene eseguito il comando all'interno della sezione per la gestione delle eccezioni relativo all'eccezione corrispondente e una riga viene inserita nella tabella AREA. Ecco l'output del blocco PL/SQL:

```

select *
from AREA;

```

| RADIUS | AREA |
|--------|-------|
| 3 | 28.27 |
| 0 | 0 |

L'output mostra che il primo valore di "Radius" (3) è stato elaborato e che l'errore è stato incontrato 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. Non è possibile utilizzare un comando `goto` per passare dalla sezione per la gestione delle eccezioni a quella dei comandi eseguibili del blocco PL/SQL. Per mantenere il controllo all'interno della sezione dei comandi eseguibili occorre utilizzare condizioni `if` per verificare la possibilità di eccezioni prima che queste vengano incontrate dal programma.

Le eccezioni definite dal sistema disponibili sono elencate alla voce "Eccezioni" nella Guida alfabetica di riferimento del Capitolo 37. Esempi di eccezioni definite dall'utente sono riportati nel Capitolo 23 e nel Capitolo 24.

• Capitolo 23

I trigger

- 23.1 **Privilegi di sistema necessari**
- 23.2 **Privilegi di tabella richiesti**
- 23.3 **Tipi di trigger**
- 23.4 **Sintassi dei trigger**
- 23.5 **Attivazione e disattivazione dei trigger**
- 23.6 **Sostituzione di trigger**
- 23.7 **Scaricamento di trigger**

Un *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 differenziale dichiarativa, per imporre complesse regole legate all’attività o per effettuare revisioni sulle modifiche ai dati. Il codice all’interno di un trigger, detto *corpo del trigger*, è costituito da blocchi PL/SQL (Capitolo 22).

L’esecuzione dei trigger è trasparente all’utente. I trigger vengono eseguiti dal database quando specifici tipi di comandi di manipolazione dei dati vengono eseguiti su specifiche tabelle. Tali comandi possono comprendere insert, update e delete. Anche gli aggiornamenti di specifiche colonne possono essere utilizzati come trigger di eventi.

Grazie alla loro flessibilità, i trigger possono aggiungere integrità referenziale, ma non possono essere utilizzati in sostituzione di questa. Per imporre le regole legate all’attività in un’applicazione è necessario in primo luogo affidarsi all’integrità referenziale dichiarativa disponibile con ORACLE. Quindi si potranno utilizzare trigger per imporre regole che non possono essere codificate attraverso l’integrità differenziale.

23.1 **Privilegi di sistema necessari**

Per creare un trigger su una tabella è necessario essere in grado di modificare la tabella stessa. Pertanto è necessario possedere la tabella, avere il privilegio ALTER per la tabella stessa o il privilegio di sistema ALTER ANY TABLE. Inoltre è necessario disporre del privilegio di sistema CREATE TRIGGER. Per creare trigger nell’account di un altro utente (altrimenti detto *schema*) è necessario 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 possederlo o disporre del privilegio di sistema ALTER ANY TRIGGER. È inoltre possibile modificare i trigger intervenendo sulle tabelle su cui essi sono basati.

Ciò richiede che si disponga del privilegio ALTER per la tabella che si desidera modificare o del privilegio di sistema ALTER ANY TABLE. Per ulteriori informazioni sulla modifica dei trigger si rimanda al paragrafo “Attivazione e disattivazione dei trigger” del presente capitolo.

23.2 Privilegi di tabella richiesti

I trigger possono fare riferimento a tabelle differenti da quella che attiva l'evento di trigger. Ad esempio, se si utilizzano trigger per revisionare modifiche ai dati nella tabella REGISTRO è possibile inserire un record in una tabella differente (ad esempio REGISTRO_AUDIT) ogni volta che un record della tabella REGISTRO è modificato.

Allo scopo è necessario disporre del privilegio di inserimento nella tabella REGISTRO_AUDIT (per effettuare la transazione con trigger).

NOTA *I privilegi necessari per le transazioni con trigger non possono derivare da ruoli. Tali privilegi devono essere concessi direttamente dal creatore del trigger.*

23.3 Tipi di trigger

Esistono 14 tipi di trigger. Un tipo di trigger è definito dal tipo di transazione e dal livello al quale il trigger viene eseguito. Nei paragrafi seguenti sono riportate le descrizioni di tali classificazioni, insieme alle corrispondenti restrizioni.

Trigger a livello di riga

I trigger a livello di riga vengono eseguiti una volta per ciascuna riga di una transazione. Per l'esempio di revisione della tabella REGISTRO descritto in precedenza, ciascuna riga modificata nella tabella REGISTRO può essere elaborata dal trigger. I trigger a livello di riga costituiscono il tipo più comune: vengono spesso utilizzati in applicazioni di revisione dei dati e si rivelano utili per mantenere sincronizzati i dati distribuiti. Gli snapshot, che utilizzano per questo scopo i trigger, sono descritti nel Capitolo 28.

Per creare trigger a livello di riga occorre utilizzare la clausola `for each row` nel comando `create trigger`. La sintassi dei trigger è riportata nel paragrafo “Sintassi dei trigger” del presente capitolo.

Trigger a livello di istruzione

I trigger *a livello di istruzione* vengono eseguiti una sola volta per ciascuna transazione. Ad esempio, se una singola transazione ha inserito 500 righe nella tabella REGISTRO, un trigger a livello di istruzione su tale tabella verrà eseguito una sola volta. I trigger a livello di istruzione non vengono pertanto utilizzati per attività correlate ai dati. Tali trigger vengono normalmente utilizzati per imporre misure aggiuntive di sicurezza sui tipi di transazione che possono essere eseguiti su una tabella.

I trigger a livello di istruzione sono il tipo predefinito creato dal comando `create trigger`. La sintassi dei trigger è riportata nel paragrafo “Sintassi dei trigger” del presente capitolo.

Trigger BEFORE e AFTER

I trigger, in quanto intervengono a causa di eventi precisi, possono essere impostati in modo da intervenire immediatamente prima o dopo tali eventi. Dato che gli eventi in grado di eseguire trigger sono transazioni di database, i trigger possono essere eseguiti immediatamente prima o dopo l'utilizzo dei comandi `insert`, `update` e `delete`.

All'interno del trigger è possibile fare riferimento ai vecchi e nuovi valori coinvolti nella transazione. L'accesso richiesto per i vecchi e i nuovi dati può determinare quale tipo di trigger sia necessario. “Vecchi” si riferisce ai dati nello stato precedente alla transazione. I comandi `update` e `delete` fanno normalmente riferimento ai vecchi valori. I valori “nuovi” sono i valori dei dati creati dalla transazione (come le colonne in un record sottoposto a `insert`).

Per impostare un valore di colonna in una riga inserita mediante un trigger, è necessario utilizzare un trigger BEFORE INSERT per accedere ai valori “nuovi”. L'utilizzo 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 frequentemente in applicazioni di revisione, in quanto non vengono attivati sino a che la riga non viene modificata. L'avvenuta modifica della riga implica che quest'ultima abbia superato con successo le restrizioni di integrità referenziale definite per quella tabella.

Trigger INSTEAD OF

In ORACLE8 è possibile utilizzare trigger INSTEAD OF per specificare che cosa fare invece di eseguire le azioni che hanno attivato il trigger. Ad esempio, è possibile utilizzare un trigger INSTEAD OF per reindirizzare gli `insert` in una tabella verso una tabella differente o per aggiornare con `update` più tabelle che siano parte di una vista. I trigger INSTEAD OF possono essere utilizzati sia su viste oggetto (Capitolo 5), sia su viste relazionali standard.

Ad esempio, se una vista comporta l'unione di due tabelle, la possibilità di utilizzare il comando `update` su record della vista è limitata. Utilizzando un trigger INSTEAD OF è possibile specificare a ORACLE come effettuare `update`, `delete` o `insert` di

record sulle tabelle sottostanti alla vista quando un utente tenta di modificare i valori attraverso la vista stessa. Il codice del trigger INSTEAD OF viene eseguito al posto del comando insert, update o delete immesso.

Nel presente capitolo viene descritta l'integrazione dei trigger di base. I trigger INSTEAD OF, inizialmente introdotti per supportare le viste oggetto, sono descritti in dettaglio nel Capitolo 25.

Tipi di trigger validi

Combinando i differenti tipi di azione di trigger si ottengono 14 configurazioni possibili.

```
BEFORE INSERT riga
BEFORE INSERT istruzione
AFTER INSERT riga
AFTER INSERT istruzione
BEFORE UPDATE riga
BEFORE UPDATE istruzione
AFTER UPDATE riga
AFTER UPDATE istruzione
BEFORE DELETE riga
BEFORE DELETE istruzione
AFTER DELETE riga
AFTER DELETE istruzione
INSTEAD OF riga
INSTEAD OF istruzione
```

I trigger UPDATE possono dipendere dalle colonne aggiornate. Per ulteriori informazioni si rimanda al paragrafo “Sintassi dei trigger” nel presente capitolo.

23.4 Sintassi dei trigger

Nell'esempio seguente è riportata la sintassi per il comando create trigger.

```
create [or replace] trigger [utente.]trigger
  {before | after | instead of}
  { delete
  | insert
  | update [of colonna [, colonna] ... ] }
    [or { delete
        | insert
        | update [of colonna [, colonna] ... ] } ] ...
on [utente.]{TABLE | VIEW}
[ [referencing { old [as] vecchio
                | new [as] nnuovo} ...]
  for each {row | statement} [when (condizione)] ] blocco pl/sql
```

Nella progettazione di un trigger è possibile una grande flessibilità. Le parole chiave before e after indicano se il trigger dovrà essere eseguito prima o dopo la transazione che attiva il trigger stesso. Con la clausola instead of il codice del trigger verrà eseguito in luogo dell'evento che ha causato l'esecuzione del trigger stesso. Le parole chiave delete, insert e update (l'ultima delle quali può comprendere un elenco di colonne) indicano il tipo di manipolazione dei dati che costituirà un evento in grado di attivare il trigger. Per fare riferimento ai vecchi e nuovi valori 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 al momento dell'esecuzione del trigger. Le restrizioni imposte per mezzo della clausola when possono comprendere verifiche dei vecchi e nuovi valori dei dati.

Si supponga, ad esempio, di voler monitorare qualsiasi variazione di un importo superiore al 10 per cento. Il trigger a livello di riga BEFORE UPDATE di seguito descritto verrà eseguito solo se il nuovo valore della colonna Importo sarà maggiore, rispetto al vecchio valore, in misura superiore al 10 per cento. L'esempio seguente descrive anche l'utilizzo della parola chiave new, che fa riferimento al nuovo valore della colonna, e di old, che fa riferimento al vecchio valore della colonna.

```
create trigger registro_bef_upd_row
before update on REGISTRO
for each row
when (new.Importo/old.Importo>1.1)
begin
insert into REGISTRO_AUDIT
values (:old.DataAzione, :old.Azione, :old.Articolo,
:old.Quantita, :old.TipoQuantita, :old.Tasso,
:old.Importo, :old.Persona);
end;
```

La suddivisione di questo comando create trigger nei suoi componenti rende più semplice la comprensione. In primo luogo viene assegnato un nome al trigger.

```
create trigger registro_bef_upd_row
```

Il nome del trigger contiene il nome della tabella su cui agisce e il tipo del trigger stesso. Per ulteriori informazioni sulle convenzioni di assegnazione dei nomi si rimanda al paragrafo "Assegnazione dei nomi ai trigger" più oltre nel presente capitolo.

```
before update on REGISTRO
```

Questo trigger si applica alla tabella REGISTRO. Il trigger verrà eseguito prima di aggiornare (before update) le transazioni trasmesse al database.

```
for each row
```

A causa dell'utilizzo della clausola for each row il trigger verrà applicato a ciascuna riga della transazione. Se tale clausola non viene utilizzata, il trigger verrà eseguito solo al livello delle istruzioni.

```
when (new.Importo/old.Importo>1.1)
```

La clausola when aggiunge ulteriori criteri alla condizione di trigger. L'evento di trigger deve non solo essere un aggiornamento (update) della tabella REGISTRO, ma anche rispecchiare un aumento superiore al 10 per cento del valore della colonna Importo.

```
begin
insert into REGISTRO_AUDIT
values (:old.DataAzione, :old.Azione, :old.Articolo,
:old.Quantita, :old.TipoQuantita, :old.Tasso,
:old.Importo, :old.Persona);
end;
```

Il codice PL/SQL dell'esempio precedente è il corpo del trigger. I comandi in esso contenuti verranno eseguiti a ogni aggiornamento (update) della tabella REGISTRO che soddisfi la condizione when. Perché ciò accada, deve esistere la tabella REGISTRO_AUDIT e il proprietario del trigger deve disporre di privilegi (diretti e non derivanti da un ruolo) rispetto a tale tabella. In questo particolare esempio si effettua l'insert dei vecchi valori dal record REGISTRO nella tabella REGISTRO_AUDIT prima che il record REGISTRO venga aggiornata.

NOTA Per il riferimento le parole chiave *new* e *old* nel blocco PL/SQL sono precedute dal segno di due punti (:).

Questo esempio è un tipico trigger di revisione. L'attività di revisione è completamente trasparente all'utente che esegue l'update della tabella REGISTRO.

In questo esempio la transazione è avvenuta all'interno della tabella REGISTRO e un'altra transazione è stata eseguita all'interno di REGISTRO_AUDIT a causa del trigger. Se si desidera che la transazione in REGISTRO_AUDIT avvenga senza inserire un record in REGISTRO, è necessario utilizzare un trigger INSTEAD OF. Il trigger INSTEAD OF verrà anch'esso creato sulla tabella REGISTRO, ma il suo codice verrà eseguito senza che venga effettuata implicitamente anche la transazione che causa la stessa esecuzione del trigger. Se si crea un trigger INSTEAD OF per gli insert nella tabella REGISTRO, tale trigger sarà in grado di inserire (insert) record in REGISTRO_AUDIT, senza inserire alcun record nella tabella REGISTRO. Per la descrizione di esempi di trigger INSTEAD OF si rimanda al Capitolo 25.

Combinazioni di trigger di tipo differente

I trigger per comandi multipli di insert, update e delete su una tabella possono essere combinati in un singolo trigger, posto che siano tutti dello stesso livello (di riga o di istruzione). Nell'esempio seguente è descritto un trigger che viene eseguito ogni volta avviene un insert o un update.

In tale esempio, due punti di particolare importanza sono indicati in grassetto: la parte update del trigger interviene solo quando la colonna Importo viene aggiornata

e all'interno del blocco PL/SQL viene utilizzata una clausola if per determinare quale dei due comandi ha eseguito il trigger.

```
create trigger registro_bef_upd_ins_row
before insert or update of Importo on REGISTRO
for each row
begin
  IF INSERTING THEN
    insert into REGISTRO_AUDIT
      values (:new.DataAzione, :new.Azione, :new.Articolo,
              :new.Quantita, :new.TipoQuantita, :new.Tasso,
              :new.Importo, :new.Persona);
  ELSE -- se non è un inserimento, si aggiorna Importo
    insert into REGISTRO_AUDIT
      values (:old.DataAzione, :old.Azione, :old.Articolo,
              :old.Quantita, :old.TipoQuantita, :old.Tasso,
              :old.Importo, :old.Persona);
  end if;
end;
```

Si può scomporre ancora il trigger nelle sue componenti. Inizialmente il trigger riceve un nome e viene identificato come trigger before insert e before update (del valore Importo), da eseguirsi per ogni riga (for each row).

```
create trigger registro_bef_upd_ins_row
before insert or update of Importo on REGISTRO
for each row
```

Segue quindi il corpo del trigger. Nella prima parte, riportata di seguito, avviene il controllo del tipo di transazione per mezzo di una clausola if. I tipi di transazione validi sono INSERTING, DELETING e UPDATING. In questo caso il trigger verifica se il record deve essere inserito nella tabella REGISTRO. 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 REGISTRO_AUDIT.

```
begin
  IF INSERTING THEN
    insert into REGISTRO_AUDIT
      values (:new.DataAzione, :new.Azione, :new.Articolo,
              :new.Quantita, :new.TipoQuantita, :new.Tasso,
              :new.Importo, :new.Persona);
```

Possono quindi essere verificati altri tipi di transazioni. In questo esempio, dal momento in cui il trigger è stato eseguito, la transazione deve essere un insert o un update della colonna Importo. Dato che la clausola if nella prima metà del corpo del trigger verifica gli insert e il trigger viene eseguito solo per insert e update, l'unica condizione in grado di eseguire la seconda metà del corpo del trigger è data da update di Importo. Per questo motivo non saranno necessarie ulteriori clausole if per determinare il tipo di transazione. Questa parte del corpo del trigger è identica a quella precedente. Prima di essere aggiornati, i vecchi valori nella riga sono scritti nella tabella REGISTRO_AUDIT.

```

ELSE -- se non è un inserimento, si aggiorna Importo
    insert into REGISTRO_AUDIT
        values (:old.DataAzione, :old.Azione, :old.Articolo,
        :old.Quantita, :old.TipoQuantita, :old.Tasso,
        :old.Importo, :old.Persona);
end;

```

La combinazione di trigger di tipo differente può agevolare la coordinazione dello sviluppo di trigger tra più sviluppatori, in quanto aiuta a consolidare tutti gli eventi di database che dipendono da una singola tabella.

Impostazione di valori inseriti

I trigger possono essere utilizzati per impostare valori di colonna durante insert e update. Ad esempio, la tabella REGISTRO può essere stata parzialmente denormalizzata per comprendere dati derivati come UPPER(Persona). La memorizzazione di questi dati in maiuscolo in una colonna della tabella (ad esempio MaiuscPersona) consente la visualizzazione dei dati agli utenti nel loro formato naturale, pur utilizzando la colonna maiuscola durante le query.

I dati MaiuscPersona, essendo derivati, possono non essere sincronizzati con la colonna Persona. Ciò significa che possono verificarsi intervalli di tempo, immediatamente dopo le transazioni, durante i quali MaiuscPersona non è uguale a UPPER(Persona). Si consideri un inserimento nella tabella REGISTRO; a meno che l'applicazione utilizzata non fornisca un valore per MaiuscPersona durante insert, il valore di quella colonna sarà NULL.

Per evitare questo problema di sincronia, è possibile utilizzare un trigger di database. Si creano un trigger BEFORE INSERT e un trigger BEFORE UPDATE per la tabella. Questi agiranno a livello di riga. Come nell'esempio seguente, i trigger imposteranno un nuovo valore per MaiuscPersona a ogni modifica di Persona.

```

create trigger registro_bef_upd_ins_row
before insert or update of Persona on REGISTRO
for each row
begin
    :new.MaiuscPersona := UPPER(:new.Persona);
end;

```

In questo esempio il corpo del trigger determina il valore per MaiuscPersona utilizzando la funzione UPPER della colonna Persona. Tale trigger verrà eseguito ogni volta che una riga viene inserita nella tabella REGISTRO e ogni volta che la colonna Persona viene aggiornata. Le due colonne verranno così mantenute in sincronia.

Conservazione di dati duplicati

Il metodo di impostazione di valori per mezzo di trigger descritto nel paragrafo precedente può essere combinato con i metodi di accesso ai dati remoti descritti nel

Capitolo 21. L'utilizzo di trigger per la duplicazione dei dati è ridondante in database dotati dell'opzione distribuita: a tale scopo vengono utilizzati gli snapshot (Capitolo 28). Se però non si dispone dell'opzione distribuita, è possibile utilizzare i trigger per conservare copie remote (sullo stesso host) delle tabelle. Come avviene per gli snapshot, è possibile replicare tutte o parte delle righe di una tabella.

Si può ad esempio desiderare di creare una copia del proprio registro di revisione dell'applicazione. In questo modo ci si cautela contro l'eliminazione, da parte di un'unica applicazione, di tutti i record del registro di revisione creati da più applicazioni. La duplicazione dei registri di revisione viene utilizzata di frequente nel monitoraggio di sicurezza.

Si consideri la tabella REGISTRO_AUDIT utilizzata nei precedenti esempi del presente capitolo. È possibile creare una seconda tabella, REGISTRO_AUDIT_DUP, possibilmente in un database remoto. Per questo esempio si assuma che un link di database denominato AUDIT_LINK possa essere utilizzato per connettere l'utente al database in cui risiede REGISTRO_AUDIT_DUP (per ulteriori informazioni sui link di database si rimanda al Capitolo 21).

Per automatizzare la popolazione della tabella REGISTRO_AUDIT_DUP, è possibile applicare alla tabella REGISTRO_AUDIT il trigger descritto di seguito.

```
create trigger registro_after_ins_row
before insert on REGISTRO_AUDIT
for each row
begin
  insert into REGISTRO_AUDIT_DUP@AUDIT_LINK
    values (:new.DataAzione, :new.Azione, :new.Articolo,
    :new.Quantita, :new.TipoQuantita, :new.Tasso,
    :new.Importo, :new.Persona);
end;
```

Come si deduce dalla sua intestazione, questo trigger viene eseguito per ciascuna riga che viene inserita nella tabella REGISTRO_AUDIT. Esso inserisce un singolo record nella tabella REGISTRO_AUDIT_DUP, nel database definito dal link AUDIT_LINK. Se si utilizza ORACLE con l'opzione distribuita, AUDIT_LINK può puntare a un database collocato su un server remoto, anche se in questo caso è consigliato l'utilizzo di uno snapshot (Capitolo 28).

Personalizzazione delle condizioni d'errore

All'interno di un singolo trigger è possibile stabilire differenti condizioni d'errore. Per ciascuna delle condizioni d'errore definite è possibile selezionare un messaggio visualizzato quando l'errore stesso interviene. I numeri degli errori e i messaggi visualizzati all'utente vengono impostati mediante la procedura RAISE_APPLICATION_ERROR. Tale procedura può essere richiamata dall'interno di qualsiasi trigger.

Nell'esempio seguente è riportato un trigger a livello di istruzione BEFORE DELETE sulla tabella REGISTRO. Quando un utente tenta il delete di un record della tabella REGISTRO, questo trigger viene eseguito e verifica due condizioni di sistema: che il giorno della settimana non sia sabato o domenica e che il nome uten-

te ORACLE dell'account che effettua l'eliminazione inizi con le lettere 'FIN'. I componenti del trigger sono descritti dopo l'esempio.

```
create trigger registro_bef_del
before delete on REGISTRO
declare
    weekend_error EXCEPTION;
    not_finance_user EXCEPTION;
begin
    IF TO_CHAR(SysDate,'DY') = 'SAT' or
        TO_CHAR(SysDate,'DY') = 'SUN' THEN
        RAISE weekend_error;
    END IF;
    IF SUBSTR(User,1,3) <> 'FIN' THEN
        RAISE not_finance_user;
EXCEPTION
    WHEN weekend_error THEN
        RAISE_APPLICATION_ERROR (-20001,
            'Eliminazioni non lecite nei fine settimana');
    WHEN not_finance_user THEN
        RAISE_APPLICATION_ERROR (-20002,
            'Eliminazioni messe solo a utenti finanziari');
end;
```

L'intestazione definisce un trigger livello di istruzione BEFORE DELETE.

```
create trigger registro_bef_del
before delete on REGISTRO
```

In questo trigger non compaiono clausole when e il corpo del trigger verrà eseguito per tutti i delete.

La successiva parte del trigger dichiara i nomi delle due eccezioni che sono definite all'interno del trigger stesso.

```
declare
    weekend_error EXCEPTION;
    not_finance_user EXCEPTION;
```

La prima parte del corpo del trigger contiene una clausola if che utilizza la funzione TO_CHAR sulla pseudocolonna SysDate. Se il giorno corrente è sabato o domenica, verrà eseguita la condizione di errore WEEKEND_ERROR. Tale condizione di errore, detta *eccezione*, deve essere definita all'interno del corpo del trigger.

```
begin
    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 'FIN'. Se il nome utente non inizia con 'FIN', verrà eseguita l'eccezione NOT_FINANCE_USER.

In questo esempio viene utilizzato l'operatore `<>`, equivalente a `!=` (che significa “non è uguale”).

```
IF SUBSTR(User,1,3) <> 'FIN' THEN
    RAISE not_finance_user;
```

La parte finale del corpo del trigger specifica come gestire le eccezioni. Tale parte inizia con la EXCEPTION, seguita da una clausola when per ciascuna delle eccezioni. Ciascuna eccezione nel trigger richiama la procedura RAISE_APPLICATION_ERROR.

La procedura RAISE_APPLICATION_ERROR riceve 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 differenti messaggi d'errore, uno per ciascuna delle eccezioni definite.

```
EXCEPTION
    WHEN weekend_error THEN
        RAISE_APPLICATION_ERROR (-20001,
            'Eliminazioni non lecite nei fine settimana');
    WHEN not_finance_user THEN
        RAISE_APPLICATION_ERROR (-20002,
            'Eliminazioni permesse solo a utenti finanziari');
END;
```

L'utilizzo della procedura RAISE_APPLICATION_ERROR consente grande flessibilità nella gestione delle condizioni di errore che possono essere incontrate all'interno del trigger. Per ulteriori informazioni sulle procedure, si rimanda al Capitolo 24.

Assegnazione dei nomi ai trigger

Il nome di un trigger deve indicare chiaramente la tabella a 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, è necessario utilizzare una serie standard di abbreviazioni. In generale il nome del trigger deve comprendere una parte quanto più possibile ampia del nome della tabella. Così, per creare un trigger a livello di riga BEFORE UPDATE sulla tabella FOGLIO_BILANCIO, il nome assegnato al trigger non dovrebbe essere BEFORE_UPDATE_LIVELLO_RIGA_FOG_BIL. Un nome migliore potrebbe essere FOGLIO_BILANCIO_BU_RIGA.

23.5 Attivazione e disattivazione dei trigger

A differenza dalle restrizioni di integrità dichiarativa (come NOT NULL e PRIMARY KEY), i trigger non hanno effetto su tutte le righe di una tabella, ma trigger influiscono solo sulle transazioni del tipo specificato e solo mentre il trigger è abilitato.

Qualsiasi transazione creata prima della creazione di un trigger non verrà influenzata dal trigger stesso.

Nel momento in cui viene creato, un trigger è per default attivato. Esistono però situazioni in cui può essere necessario disattivare un trigger. Le due ragioni più comuni riguardano il caricamento dei dati. Durante grandi caricamenti di dati possono essere disattivati i trigger che verrebbero eseguiti durante il caricamento stesso; ciò può migliorare enormemente le prestazioni. Una volta completato il caricamento dei dati, sarà necessario eseguire manualmente il trattamento dei dati che sarebbe stato effettuato dal trigger se questo fosse stato attivo.

La seconda ragione per la disattivazione di un trigger, correlata al caricamento dei dati, interviene quando un caricamento di dati non viene completato con successo e deve essere effettuato una seconda volta. In tal caso è probabile che il caricamento dei dati abbia avuto successo parziale e che il trigger sia stato eseguito per una parte dei record interessati. Durante un successivo caricamento dei dati, verrebbero inseriti gli stessi record. Così, è possibile che lo stesso trigger venga eseguito due volte per la stessa transazione (quando tale transazione interviene durante ambedue i caricamenti).

A seconda della natura della transazione e dei trigger ciò può risultare errato. Se il trigger era attivato durante il caricamento non completato con successo, sarà necessario disattivarlo prima di avviare un secondo processo di caricamento dei dati. Una volta che i dati sono stati caricati, sarà necessario eseguire manualmente il trattamento dei dati che sarebbe stato effettuato dal trigger se questo fosse stato attivo durante il caricamento dei dati stessi.

Per attivare un trigger si utilizza il comando `alter trigger` con la parola chiave `enable`. Per utilizzare tale comando è necessario possedere la tabella o disporre del privilegio di sistema `ALTER ANY TRIGGER`. Nell'esempio seguente è riportato un modello di comando `alter trigger`.

```
alter trigger registro_bef_upd_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 (per tale scopo è necessario utilizzare il comando `alter trigger`). Nell'esempio seguente è descritto l'utilizzo del comando `alter table`:

```
alter table REGISTRO enable all triggers;
```

Per utilizzare il comando `alter table` è necessario possedere la tabella o disporre del privilegio di sistema `ALTER ANY TABLE`.

I trigger possono essere disattivati utilizzando gli stessi comandi di base (e con necessità di disporre degli stessi privilegi) con clausole modificate. Per il comando `alter trigger` si utilizza la clausola `disable`.

```
alter trigger registro_bef_upd_row disable;
```

Per il comando `alter table` si utilizza utilizzare la clausola `disable all triggers`, come nell'esempio seguente:

```
alter table REGISTRO disable all triggers;
```

A partire da ORACLE7.3 è possibile compilare i trigger. Per compilare manualmente trigger esistenti che siano divenuti non validi si utilizza il comando `alter trigger compile`. I trigger, che a partire da ORACLE7.3 hanno delle dipendenze, possono divenire non validi se l'oggetto da cui dipendono viene modificato. Il comando `alter trigger debug`, disponibile anch'esso a partire da ORACLE7.3, consente la generazione di informazioni PL/SQL durante la ricompilazione dei trigger.

23.6 Sostituzione di trigger

Il corpo di un trigger non può essere modificato. Come è stato spiegato nei paragrafi precedenti, lo stato di un trigger è l'unica parte di esso che può essere modificata.

Per modificare il corpo di un trigger è necessario ricreare o sostituire il trigger stesso.

Per la sostituzione di un trigger si utilizza il comando `create or replace trigger` (descritto nel precedente paragrafo “Sintassi dei trigger”).

23.7 Scaricamento di trigger

I trigger possono essere scaricati mediante il comando `drop trigger`. Per scaricare un trigger è necessario possedere il trigger stesso o disporre del privilegio di sistema `DROP ANY TRIGGER`. Di seguito è riportato un esempio di questo comando:

```
drop trigger registro_bef_upd_row;
```

Capitolo 24

Le procedure

- 24.1 **Privilegi di sistema necessari**
- 24.2 **Privilegi di tabella necessari**
- 24.3 **Procedure e funzioni**
- 24.4 **Procedure e package**
- 24.5 **Sintassi del comando create procedure**
- 24.6 **Sintassi del comando create function**
- 24.7 **Sintassi per il comando create package**
- 24.8 **Visualizzazione del codice sorgente di oggetti procedurali esistenti**
- 24.9 **Compilazione di procedure, funzioni e package**
- 24.10 **Sostituzione di procedure, funzioni e package**
- 24.11 **Scaricamento di procedure, funzioni e package**

In ORACLE sofisticate regole di attività e logiche di applicazione possono essere memorizzate come *procedure*. Le *procedure memorizzate*, gruppi di istruzioni SQL e PL/SQL, consentono lo spostamento di 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. Grazie al fatto che ORACLE supporta le procedure memorizzate, il codice all'interno delle applicazioni diviene più coerente e di facile manutenzione.

Le procedure e gli altri comandi PL/SQL possono essere raggruppati in *package*. Nei paragrafi successivi sono descritti dettagli di integrazione e raccomandazioni riguardanti i package, le procedure e le *funzioni* (procedure che possono restituire valori all'utente).

L'utilizzo di procedure può assicurare miglioramenti delle prestazioni per due ragioni. In primo luogo l'elaborazione di complesse regole legate all'attività può essere effettuata all'interno del database e quindi mediante il server. Nelle applicazioni client-server lo spostamento di elaborazioni complesse dall'applicazione (sul client) al database (sul server) può migliorare sensibilmente le prestazioni. In secondo luogo, dato che il codice delle procedure è memorizzato all'interno del database ed è piuttosto statico, è anche possibile beneficiare del riutilizzo delle stesse

query all'interno del database. L'area SQL condivisa (Shared SQL Area) dell'area globale di sistema (System Global Area) memorizza le versioni parcellizzate dei comandi eseguiti. Così una procedura, la seconda volta che viene eseguita, può avvantaggiarsi della parcellizzazione effettuata in precedenza, migliorando le prestazioni nell'esecuzione.

Oltre a ciò, anche lo sviluppo di applicazioni può essere avvantaggiato. Se si consolidano le regole legate all'attività all'interno del database, queste non dovranno più essere scritte in ciascuna applicazione, il che consente risparmi di tempo durante la creazione delle applicazioni stesse e semplifica le procedure di manutenzione.

24.1 **Privilegi di sistema necessari**

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 dovrà trovare 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. Per l'esecuzione un oggetto procedurale fa riferimento ai privilegi di tabella del suo proprietario, non ai privilegi dell'utente che esegue l'oggetto stesso. Per eseguire una procedura, un utente non deve obbligatoriamente disporre del privilegio di accesso alle tabelle cui accede la procedura stessa.

Per consentire ad altri utenti di eseguire un oggetto procedurale, occorre utilizzare il comando grant per il privilegio EXECUTE come nell'esempio seguente.

```
grant execute on MIA_PROCEDURA to Dora;
```

L'utente Dora sarà in grado di eseguire la procedura MIA_PROCEDURA pur non disponendo di privilegi su nessuna delle tabelle utilizzate da questa. Se non si effettua un grant per il privilegio EXECUTE a favore di altri utenti, questi dovranno disporre, per eseguire la procedura, del privilegio di sistema EXECUTE ANY PROCEDURE.

Durante l'esecuzione delle procedure, normalmente a queste ultime vengono passate delle variabili. Ad esempio, una procedura che interagisca con la tabella REGISTRO potrà accettare come dato in input un valore per la colonna Persona. In questo esempio si suppone che la procedura crei un nuovo record nella tabella LAVORATORE quando un nuovo dipendente viene pagato per la prima volta (causando l'aggiunta di una voce in REGISTRO). Questa procedura può essere richiamata da qualsiasi applicazione all'interno del database (posto che l'utente che richiama la procedura disponga del privilegio EXECUTE per essa).

La sintassi utilizzata per eseguire una procedura dipende dall'ambiente da cui la procedura viene richiamata. Dall'interno di SQLPLUS una procedura può essere eseguita utilizzando il comando execute, seguito dal nome della procedura stessa.

Qualsiasi argomento da passare alla procedura deve essere racchiuso tra parentesi e seguire il nome della procedura, come nell'esempio seguente (che utilizza la procedura NUOVO_LAVORATORE).

```
execute NUOVO_LAVORATORE('ADAH TALBOT');
```

Il comando dell'esempio precedente esegue la procedura NUOVO_LAVORATORE passando a quest'ultima il valore 'ADAH TALBOT'.

Dall'interno di un'altra procedura, funzione, package o trigger, la procedura può essere richiamata senza il comando execute.

Se la procedura NUOVO_LAVORATORE è stata richiamata da un trigger sulla tabella REGISTRO, il corpo di detto trigger potrà comprendere il comando:

```
NUOVO_LAVORATORE(:new.Persona);
```

In questo esempio la procedura NUOVO_LAVORATORE verrà eseguita utilizzando come dato in input il nuovo valore della colonna Persona. Per ulteriori informazioni sull'utilizzo di valori vecchi e nuovi all'interno dei trigger si rimanda al Capitolo 23.

Per eseguire una procedura posseduta da un altro utente è necessario creare un sinonimo per tale procedura o fare riferimento al nome del proprietario durante l'esecuzione, come nell'esempio seguente:

```
execute Dora.NUOVO_LAVORATORE('ADAH TALBOT');
```

Il comando dell'esempio precedente esegue la procedura NUOVO_LAVORATORE di proprietà di Dora. In alternativa, è possibile creare un sinonimo per la procedura come nell'esempio seguente:

```
create synonym NUOVO_LAVORATORE for Dora.NUOVO_LAVORATORE;
```

Il proprietario di tale sinonimo non dovrà più fare riferimento al proprietario della procedura per eseguire la stessa. Sarà sufficiente immettere il comando descritto di seguito:

```
execute NUOVO_LAVORATORE('ADAH TALBOT');
```

e il sinonimo punterà alla corretta procedura.

Per l'esecuzione di procedure remote è necessario specificare il nome di un link di database (per ulteriori informazioni sui link di database si rimanda al Capitolo 21). Il nome del link deve essere specificato dopo il nome della procedura ma prima delle variabili, come nell'esempio seguente:

```
execute NUOVO_LAVORATORE@REMOTE_CONNECT('ADAH TALBOT');
```

Questo comando utilizza il link di database REMOTE_CONNECT per accedere a una procedura di nome NUOVO_LAVORATORE in un database remoto. Per rendere l'individuazione della procedura trasparente all'utente si può creare un sinonimo per la procedura remota, come nell'esempio seguente.

```
create synonym NUOVO_LAVORATORE
for NUOVO_LAVORATORE@REMOTE_CONNECT;
```

Una volta creato il sinonimo, l'utente potrà fare riferimento alla procedura remota utilizzando il nome del sinonimo stesso. ORACLE assume che tutte le chiamate a procedure remote comportino aggiornamenti nei database remoti. Per questo motivo, per richiamare procedure remote è necessario che l'opzione distribuita sia installata.

24.2 Privilegi di tabella necessari

Gli oggetti procedurali possono fare riferimento a tabelle. Perché gli oggetti vengano eseguiti correttamente, il proprietario della procedura, del package o della funzione da eseguire deve disporre di privilegi sulle tabelle utilizzate. L'utente che eseguirà l'oggetto procedurale non dovrà disporre di privilegi sulle tabelle da esso utilizzate. Tale utente dovrà disporre solamente del privilegio EXECUTE sull'oggetto procedurale stesso.

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.*

24.3 Procedure e funzioni

A differenza delle procedure, le funzioni possono restituire un valore al chiamante (le procedure non sono in grado di restituire valori). Tale 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 oltre nel presente capitolo.

24.4 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 è necessario riportare prima il nome del package e quindi il nome della procedura, come nell'esempio seguente.

```
execute REGISTRO_PACKAGE.NUOVO_LAVORATORE('ADAH TALBOT');
```

In questo caso viene eseguita la procedura NUOVO_LAVORATORE all'interno del package REGISTRO_PACKAGE.

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_LAVORATORE nell'esempio precedente) o private, nel qual caso sarà possibile accedervi solo mediante comandi all'interno del packa-

ge (come chiamate da altre procedure). Esempi di package sono descritti nel paragrafo “Sintassi del comando create package”, più oltre nel presente capitolo.

Le procedure possono anche comprendere comandi da eseguire ognqualvolta il package viene richiamato, a prescindere dalla procedura o funzione richiamata all'interno del package. Così un package non solo raggruppa procedure, ma consente anche l'esecuzione di comandi che non siano specifici delle procedure raggruppate. Esempi di codice eseguito ognqualvolta un package viene richiamato sono riportati nel paragrafo “Inizializzazione dei package” più oltre nel presente capitolo.

24.5 Sintassi del comando create procedure

Di seguito è riportata la sintassi per il comando create procedure.

```
create [or replace] procedure [utente.] procedura
[(argomento [IN|OUT|IN OUT] tipodati
[,argomento [IN|OUT|IN OUT] tipodati]...)]
{IS|AS} {blocco | programma esterno};
```

Con tale comando vengono creati sia il titolo, sia il corpo della procedura. Di seguito è riportato il comando per la creazione della procedura NUOVO_LAVORATORE.

```
create procedure NUOVO_LAVORATORE (Nome_Persona IN varchar2)
AS
BEGIN
    insert into LAVORATORE
        (Nome, Eta, Alloggio)
    values
        (Nome_Persona, null, null);
END;
/
```

La procedura NUOVO_LAVORATORE in questo esempio accetta come dato in input un nome di persona e può essere richiamata da qualsiasi applicazione. Tale procedura inserisce un record nella tabella LAVORATORE con valori NULL per le colonne Eta e Alloggio.

È possibile sostituire una procedura esistente mediante il comando `create or replace procedure`. Il vantaggio nell'utilizzare tale comando invece di eliminare e ricreare la procedura consiste nel fatto che i privilegi EXECUTE esistenti relativi alla procedura resteranno invariati.

Per gli argomenti per i quali è necessario specificare un valore quando la procedura viene richiamata, si utilizza il qualificatore IN. Negli esempi relativi a NUOVO_LAVORATORE riportati in precedenza nel presente capitolo, l'argomento Persona dovrebbe essere dichiarato come IN.

Il qualificatore OUT significa che la procedura passa indietro al chiamante un valore attraverso l'argomento in questione.

Il qualificatore IN OUT significa che l'argomento è sia IN, sia OUT: quando la procedura viene richiamata 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.

blocco indica il blocco PL/SQL che la procedura eseguirà una volta richiamata. Di seguito è riportato il blocco relativo all'esempio NUOVO_LAVORATORE.

```
BEGIN
    insert into LAVORATORE
        (Nome, Eta, Alloggio)
    values
        (Nome_Persona, null, null);
END;
```

Questo blocco PL/SQL è piuttosto semplice, in quanto consiste in una singola 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 ORACLE8 è possibile richiamare procedure esterne mediante il comando create procedure. Una procedura esterna è una parte di un programma memorizzato all'esterno del database ORACLE (come un programma in C). Di seguito è riportata la sintassi per la clausola external program nel comando create procedure.

```
external library [utente.]nome_libreria
    [name nome_procedura_esterna]
    [language nome_linguaggio]
    [calling standard [c | pascal]]
parameters (elenco_parametri_esterni) [with context]
```

La libreria deve essere stata creata in precedenza mediante il comando create library. Questo comando assegna un nome a una libreria condivisa nel sistema operativo. Nell'esempio seguente è descritta la creazione di una libreria di nome MIA_LIB, che punta a una libreria di nome "/ora/progs.so".

```
create library MIA_LIB as '/ora/progs.so';
```

Una volta creata la libreria sarà possibile utilizzare i suoi programmi come parte del comando create procedure. Allo scopo sarà necessario disporre del privilegio EXECUTE sulla libreria.

24.6 Sintassi del comando create function

Di seguito è riportata la sintassi per il comando create function, molto simile alla sintassi per il comando create procedure.

```
create [or replace] function [utente.]funzione
[(argomento [IN|OUT|IN OUT] tipodati
[,argomento [IN|OUT|IN OUT] tipodati]...)]
RETURN tipodati
{IS|AS} {blocco | corpo_esterno};
```

Con tale comando vengono creati sia il titolo, sia il corpo della procedura.

La parola chiave return specifica il tipo di dati del valore restituito dalla funzione. Questo può essere qualsiasi tipo di dati valido nel linguaggio PL/SQL. Ogni funzione deve comprendere una clausola return, in quanto una funzione deve per definizione restituire un valore all'ambiente che la richiama.

Di seguito è descritta la funzione CONTROLLO_SALDO. Tale funzione restituisce lo stato delle transazioni ‘COMPRATO’ e ‘VENDUTO’ per una Persona nella tabella REGISTRO. Il dato in input è il nome della Persona mentre il dato in output è il saldo relativo alla stessa persona.

```
create function CONTROLLO_SALDO (Nome_Persona IN varchar2)
  RETURN NUMBER
  IS
    saldo NUMBER(10,2);
  BEGIN
    select sum(decode(Azione,'COMPRATO',Importo,0))
      - sum(decode(Azione,'VENDUTO',Importo,0))
    INTO saldo
    from REGISTRO
    where Persona = Nome_Persona;
    RETURN(saldo);
  END;
/
```

Per analizzare la differenza tra procedure e funzioni, si esamina la funzione pezzo per pezzo. Per prima cosa viene assegnato un nome alla funzione. Viene quindi specificato il dato in input.

```
create function CONTROLLO_SALDO (Nome_Persona IN varchar2)
```

In seguito vengono definite le caratteristiche del valore da restituire. La definizione della variabile il cui valore deve essere restituito è composta di tre parti: il tipo di dati, il nome e la lunghezza. Il tipo di dati in questo esempio viene stabilito mediante la clausola return number. Alla variabile viene quindi assegnato il nome “saldo” e la variabile stessa viene definita in questo caso come NUMBER(10,2). In questo modo sono state definite tutte le tre 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 la somma di tutti gli importi ‘VENDUTO’ viene sottratta dalla somma di tutti gli importi ‘COMPRATO’ per la Persona. La differenza tra tali somme è selezionata in una variabile di nome “saldo” (come definita in precedenza). Il comando RETURN(saldo) restituisce quindi il valore della variabile “saldo” al programma chiamante.

```
BEGIN
  select sum(decode(Azione,'COMPRATO',Importo,0))
    - sum(decode(Azione,'VENDUTO',Importo,0))
  INTO saldo
```

```

from REGISTRO
where Persona = Nome_Persona;
RETURN(saldo);
END;
/

```

È possibile sostituire una funzione esistente mediante il comando `create or replace function`. Utilizzando la clausola `or replace`, qualsiasi privilegio `EXECUTE` preventivamente concesso alla funzione resterà invariato.

Per creare funzione in un account differente (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.

Riferimenti a tabelle remote nelle procedure

L'accesso a tabelle remote è consentito dalle istruzioni SQL nelle procedure. È possibile effettuare una query su una tabella remota mediante un link di database nella procedura, come nell'esempio seguente, dove la procedura `NUOVO_LAVORATORE` inserisce un record nella tabella `LAVORATORE` del database definito dal link `REMOTE_CONNECT`.

```

create or replace procedure NUOVO_LAVORATORE
  (Nome_Persona IN varchar2)
AS
BEGIN
  insert into LAVORATORE@REMOTE_CONNECT
    (Nome, Eta, Alloggio)
  values
    (Nome_Persona, null, null);
END;
/

```

La manipolazione dei dati nei database remoti (come il comando `insert` nell'esempio precedente) richiede normalmente che l'opzione distribuita sia installata nel database utilizzato.

Le procedure possono inoltre utilizzare sinonimi locali. Ad esempio, è possibile creare un sinonimo locale per una tabella remota, come nell'esempio seguente:

```
create synonym LAVORATORE for LAVORATORE@REMOTE_CONNECT;
```

Sarà quindi possibile riscrivere la procedura per rimuovere la specificazione del link di database.

```

create or replace procedure NUOVO_LAVORATORE
  (Nome_Persona IN varchar2)
AS
BEGIN
  insert into LAVORATORE

```

```

        (Nome, Eta, Alloggio)
values
        (Nome_Persona, null, null);
END;
/

```

La rimozione dei nomi di link da una procedura 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 con l'oggetto procedurale creato per ultimo. Tale comando verifica nella vista del dizionario di dati USER_ERRORS gli errori associati con il più recente tentativo di compilazione per l'oggetto procedurale in esame. Il comando show errors visualizza il numero di riga e di colonna per ciascun errore, insieme al testo del messaggio di errore.

Per visualizzare errori associati con oggetti procedurali creati in precedenza, è possibile utilizzare direttamente una query su USER_ERRORS, come nell'esempio seguente. Le query su USER_ERRORS non sono comuni, in quanto implicano la presenza di errori in due o più procedure. Nell'esempio seguente è descritta una query su USER_ERRORS relativa ai messaggi d'errore incontrati durante la creazione della funzione CONTROLLO_SALDO descritta in precedenza nel capitolo.

```

select Line,      /*Numero di riga dell'errore.*/
       Position, /*Numero di colonna dell'errore.*/
       Text      /*Testo del messaggio di errore.*/
  from USER_ERRORS
 where Name = 'CONTROLLO_SALDO'
   and Type = 'FUNCTION'
 order by Sequence;

```

I valori validi per la colonna Type sono PROCEDURE, PACKAGE, FUNCTION e PACKAGE BODY.

Per il prelevamento di informazioni su errori relativi a oggetti procedurali è possibile utilizzare anche due ulteriori viste del dizionario dati, ALL e DBA. Per ulteriori informazioni su tali viste si rimanda al Capitolo 32.

Utilizzo del package DBMS_OUTPUT Oltre alle informazioni per il debugging messe a disposizione dal comando show errors, è possibile utilizzare il package DBMS_OUTPUT, creato quando l'opzione procedurale viene installata nel database.

Di seguito è riportato il comando da immettere, per utilizzare DBMS_OUTPUT, prima di eseguire l'oggetto procedurale di cui si desidera effettuare il debugging.

```
set serveroutput on
```

Il package DBMS_OUTPUT consente di utilizzare le tre funzioni di debugging elencate di seguito.

| | |
|----------|--|
| PUT | Colloca più output sulla stessa riga. |
| PUT_LINE | Colloca ciascuna uscita su una riga separata. |
| NEW_LINE | Utilizzata con PUT; segnala la fine della riga di output corrente. |

PUT e PUT_LINE vengono utilizzate per generare le informazioni di debugging che si desidera visualizzare. Se ad esempio si effettua il debugging di una procedura che comprende un ciclo (Capitolo 22), è possibile tenere traccia delle modifiche in una variabile a ogni successiva esecuzione del ciclo in questione. Allo scopo è possibile utilizzare un comando simile a quello descritto nel precedente esempio. In questo esempio viene stampato il valore della colonna Importo preceduto dalla stringa di caratteri 'Importo:'.

```
PUT_LINE('Importo: '||Importo);
```

PUT e PUT_LINE possono essere utilizzate anche al di fuori dei cicli, ma gli scopi sottesi a tali utilizzi possono essere raggiunti in modo migliore per mezzo del comando RETURN nelle funzioni (si rimanda al paragrafo "Sintassi per il comando create function" riportato in precedenza nel presente capitolo).

Creazione di funzioni personalizzate

A partire da ORACLE7.1 è possibile utilizzare funzioni personalizzate create all'interno di espressioni SQL (invece di richiamarle semplicemente per mezzo di un comando execute). Ciò consente di ampliare la funzionalità del linguaggio SQL, personalizzandolo sulle proprie esigenze. Le funzioni personalizzate possono essere utilizzate nello stesso modo in cui si utilizzano le funzioni messe a disposizione da ORACLE come SUBSTR e TO_CHAR. L'unica limitazione alla loro applicazione consiste nel fatto che le funzioni personalizzate non possono essere utilizzate in delimitatori CHECK o DEFAULT.

Le funzioni che possono essere richiamate devono essere autonome (create per mezzo del comando create function descritto nei paragrafi precedenti) o dichiarate nelle specifiche di package (si rimanda al paragrafo "Sintassi per il comando create package" più oltre nel presente capitolo).

Le procedure non possono essere richiamate direttamente da SQL, ma possono essere richiamate da funzioni personalizzate.

Si consideri ad esempio la funzione CONTROLLO_SALDO descritta in precedenza nel presente capitolo. La funzione CONTROLLO_SALDO calcolava la differenza tra i saldi 'COMPRATO' e 'VENDUTO' per diverse persone. Tale funzione aveva una singola variabile in input: il nome della persona interessata. Così, per visualizzare il risultato della funzione CONTROLLO_SALDO per tutti i lavoratori, normalmente sarebbe stato necessario eseguire la procedura una volta per ciascun record contenuto nella tabella LAVORATORE.

Il procedimento di calcolo della funzione CONTROLLO_SALDO può essere migliorato. Si consideri la query riportata di seguito.

```
select Name,
       CONTROLLO_SALDO(Nome)
  from LAVORATORE;
```

Questa singola query utilizza la funzione personalizzata CONTROLLO_SALDO per calcolare la differenza tra i saldi ‘COMPRATO’ e ‘VENDUTO’ per tutti i lavoratori.

Di seguito è riportato il risultato della query.

| NOME | CONTROLLO_SALDO(NOME) |
|---------------------------|-----------------------|
| BART SARJEANT | 0 |
| ELBERT TALBOT | 0 |
| DONALD ROLLO | 0 |
| JED HOPKINS | 0 |
| WILLIAM SWING | 0 |
| JOHN PEARSON | -.96 |
| GEORGE OSCAR | 4.5 |
| KAY AND PALMER WALLBOM | 0 |
| PAT LAVAY | -35.15 |
| RICHARD KOCH AND BROTHERS | 0 |
| DICK JONES | 0 |
| ADAH TALBOT | -7 |
| ROLAND BRANDT | -6.96 |
| PETER LAWSON | 6.5 |
| VICTORIA LYNN | -.8 |
| WILFRED LOWELL | 0 |
| HELEN BRANDT | 6.75 |
| GERHARDT KENTGEN | -9.23 |
| ANDREW DYE | 46.75 |

19 rows selected.

La query ha selezionato ciascun nome della tabella LAVORATORE e ha eseguito la funzione CONTROLLO_SALDO per ciascun nome, con i dati selezionati di REGISTRO.

Per avvantaggiarsi di questa caratteristica, le funzioni devono seguire le stesse direttive delle funzioni di ORACLE. In particolare, non devono aggiornare il database e devono contenere solo parametri IN.

Personalizzazione di condizioni di errore

All’interno degli oggetti procedurali è possibile stabilire differenti condizioni di errore (esempi di condizioni di errore personalizzate all’interno di trigger sono riportati nel Capitolo 23). Per ciascuna delle condizioni di errore definite sarà possibile scegliere un messaggio di errore, che verrà visualizzato al verificarsi dell’errore stesso. I numeri degli errori e i messaggi visualizzati all’utente possono essere impostati

mediante la procedura RAISE_APPLICATION_ERROR, che può essere richiamata dall'interno di qualsiasi oggetto procedurale.

La procedura RAISE_APPLICATION_ERROR può essere richiamata dall'interno di procedure, package e funzioni. Essa richiede due dati in input: il numero del messaggio e il testo.

Sia il numero sia il testo del messaggio che verrà visualizzato all'utente possono essere impostati. Questa rappresenta un'aggiunta molto potente alle eccezioni standard disponibili nel linguaggio PL/SQL (si consulti il paragrafo dedicato alle eccezioni nella guida di riferimento alfabetica del Capitolo 37).

Nell'esempio seguente è riportata la funzione CONTROLLO_SALDO descritta in precedenza nel presente capitolo. Ora però essa contiene una sezione aggiuntiva (evidenziata in **grassetto**), con titolo EXCEPTION, che specifica a ORACLE come gestire le elaborazioni non standard.

In questo esempio, il messaggio standard di eccezione NO_DATA_FOUND viene ri-definito per mezzo della procedura RAISE_APPLICATION_ERROR.

```
create function CONTROLLO_SALDO (Nome_Persona IN varchar2)
  RETURN NUMBER
  IS
    saldo NUMBER(10,2);
  BEGIN
    select sum(decode(Azione,'COMPRATO',Importo,0))
      - sum(decode(Azione,'VENDUTO',Importo,0))
    INTO saldo
    from REGISTRO
    where Persona = Nome_Persona;
  RETURN(saldo);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR (-20100,
        'Nessuna voce COMPRATO o VENDUTO per la Persona indicata.');
  END;
  /

```

Nell'esempio precedente viene utilizzata l'eccezione NO_DATA_FOUND. Per definire eccezioni personalizzate è necessario assegnare un nome all'eccezione in una sezione DECLARE della procedura. La sezione DECLARE precede immediatamente il comando begin e deve comprendere delle voci per ciascuna delle eccezioni personalizzate definite, elencate come tipo "EXCEPTION", come nell'esempio seguente.

```
declare
  un_errore_personalizzato EXCEPTION;
```

NOTA Se si utilizzano le eccezioni già definite all'interno del linguaggio PL/SQL, non è necessario elencarle nella sezione declare dell'oggetto procedurale. Per un elenco delle eccezioni predefinite si rimanda al paragrafo dedicato alle eccezioni della guida di riferimento alfabetica del Capitolo 37.

Nella parte exception del codice relativo all'oggetto procedurale si specifica al database come gestire le eccezioni. Questa parte inizia con la parola chiave excep-

tion, seguita da una clausola WHEN per ciascuna delle eccezioni. Ogni eccezione è in grado di richiamare la procedura RAISE_APPLICATION_ERROR.

La procedura RAISE_APPLICATION_ERROR richiede due parametri in input: il numero dell'errore (che deve essere compreso tra -20001 e -20999) e il messaggio d'errore da visualizzare. Nell'esempio precedente era definita una sola eccezione. È però possibile definire più eccezioni, come nell'esempio seguente. Allo scopo è possibile utilizzare la clausola when others per gestire tutte le eccezioni non specificate.

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR (-20100,
      'Nessuna voce COMPRATO o VENDUTO per la Persona indicata.');
  WHEN un_errore_personalizzato THEN
    RAISE_APPLICATION_ERROR (-20101,
      'Un messaggio di errore personalizzato.');
```

L'utilizzo della procedura RAISE_APPLICATION_ERROR conferisce grande flessibilità nella gestione delle condizioni di errore che si possono incontrare all'interno di oggetti procedurali.

Assegnazione di nomi alle procedure

Alle procedure e alle funzioni dovrebbe essere assegnato un nome coerente con la funzione da esse svolta o dalle regole da esse imposte. A questo scopo è necessario evitare le ambiguità.

Alla procedura NUOVO_LAVORATORE descritta in precedenza è necessario assegnare un nuovo nome. La procedura NUOVO_LAVORATORE svolge una funzione gestionale: l'inserimento di record nella tabella LAVORATORE. Il suo nome dovrebbe pertanto rispecchiare tale funzione. Un nome migliore potrebbe essere AGGIUNGENUOVO_LAVORATORE. Dato che la procedura esegue una funzione, un verbo (in questo caso "aggiunge") descrive la funzione eseguita dalla procedura. Il nome della procedura dovrebbe inoltre contenere il nome delle principali tabelle su cui agisce. Un nome adeguato dovrebbe essere quello dell'oggetto su cui agisce direttamente il verbo (in questo caso LAVORATORE). Per ulteriori informazioni sui nomi degli oggetti si rimanda al Capitolo 35.

Per motivi di coerenza si continua a riferirsi alla procedura con il nome NUOVO_LAVORATORE per la restante parte del presente capitolo.

24.7 Sintassi per il comando create package

Per la creazione di package, la specifica del package e il corpo del package stesso 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.

Ambedue tali comandi richiedono il privilegio di sistema CREATE PROCEDURE. Se il package deve essere creato nello schema di un utente differente, e non nel proprio, è necessario disporre del privilegio di sistema CREATE ANY PROCEDURE.

Di seguito è riportata la sintassi per la creazione di specifiche di package.

```
create [or replace] package [utente.] package
{IS | AS}
specifica di package PL/SQL;
```

Una *specifica di package* consiste nell'elenco delle funzioni, procedure, variabili, costanti, cursori ed eccezioni che saranno disponibili all'utente del package.

Di seguito è riportato un esempio di utilizzo del comando create package. In questo esempio viene creato il package REGISTRO_PACKAGE. Sono visualizzate la funzione CONTROLLO_SALDO e la procedura NUOVO_LAVORATORE descritte in precedenza nel presente capitolo.

```
create package REGISTRO_PACKAGE
AS
function CONTROLLO_SALDO(Nome_Persona VARCHAR2);
procedure NUOVO_LAVORATORE(Nome_Persona IN VARCHAR2);
end REGISTRO_PACKAGE;
```

NOTA Il nome dell'oggetto procedurale può essere aggiunto alla clausola *end*, come nell'esempio precedente. Questa aggiunta rende più agevole la coordinazione della logica all'interno del proprio codice.

Il *corpo di un package* contiene i blocchi e le specificazioni PL/SQL 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. Il corpo di un package può anche comprendere del codice che viene eseguito ogniqualvolta il package viene richiamato, a prescindere dalla parte del package in corso d'esecuzione (per un esempio, si rimanda al paragrafo "Inizializzazione di package" più avanti in questo capitolo).

Di seguito è riportata la sintassi per la creazione dei corpi dei package.

```
create [or replace] package body [user.] corpo package
{IS | AS}
corpo package PL/SQL;
```

Il nome del corpo del package dovrebbe coincidere con quello delle specifiche. Continuando nell'esempio di REGISTRO_PACKAGE, il corpo del package può essere creato mediante il comando create package body riportato nell'esempio seguente.

```
create package body REGISTRO_PACKAGE
AS
function CONTROLLO_SALDO (Nome_Persona IN varchar2)
RETURN NUMBER
IS
saldo NUMBER(10,2);
BEGIN
```

```

select sum(decode(Azione,'COMPRATO',Importo,0))
  - select sum(decode(Azione,'VENDUTO',Importo,0))
  INTO saldo
  from REGISTRO
  where Persona = Nome_Persona;
RETURN(saldo);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR (-20100,
      'Nessuna voce COMPRATO o VENDUTO per la Persona indicata.');
END CONTROLLO_SALDO;
procedure NUOVO_LAVORATORE
  (Nome_Persona IN varchar2)
AS
BEGIN
  insert into LAVORATORE
    (Nome, Eta, Alloggio)
  values
    (Nome_Persona, null, null);
END NUOVO_LAVORATORE;
END REGISTRO_PACKAGE;
/

```

Il comando create package body nell'esempio precedente combina il comando create function per la funzione CONTROLLO_SALDO con il comando create procedure per la procedura NUOVO_LAVORATORE. Le clausole end hanno tutte il nome degli oggetti a esse associati e aggiunti (riportati in **grassetto** nell'esempio precedente). La modifica in questo modo delle clausole end contribuisce a chiarire i punti esterni del codice dell'oggetto.

All'interno del corpo del package è possibile definire funzioni aggiuntive, procedure, eccezioni, variabili, cursori e costanti, che non saranno però disponibili per il pubblico a meno che non vengano dichiarate all'interno delle specifiche del package (per mezzo del comando create package). Se un utente dispone del privilegio EXECUTE su un package, sarà in grado di accedere a tutti gli oggetti pubblici dichiarati nelle specifiche del package stesso.

Inizializzazione dei package

I package possono comprendere codice che dovrà essere eseguito la prima volta che un utente esegue il package. Nell'esempio seguente, il corpo del package REGISTRO_PACKAGE è modificato in modo da comprendere un'istruzione SQL che registra il nome dell'utente corrente e data e ora di avvio dell'esecuzione del package. Per registrare tali valori devono inoltre essere dichiarate due nuove variabili nel corpo del package.

Le due variabili, essendo 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.

```

create or replace package body REGISTRO_PACKAGE
AS
User_Name VARCHAR2;
Entry_Date DATE;
function CONTROLLO_SALDO (Nome_Persona IN varchar2)
    RETURN NUMBER
IS
    saldo NUMBER(10,2);
BEGIN
    select sum(decode(Azione,'COMPRATO',Importo,0))
        - select sum(decode(Azione,'VENDUTO',Importo,0))
        INTO saldo
        from REGISTRO
        where Persona = Nome_Persona;
    RETURN(saldo);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR (-20100,
            'Nessuna voce COMPRATO o VENDUTO per la Persona indicata.');
    END CONTROLLO_SALDO;
procedure NUOVO_LAVORATORE
    (Nome_Persona IN varchar2)
AS
BEGIN
    insert into LAVORATORE
        (Nome, Eta, Alloggio)
    values
        (Nome_Persona, null, null);
    END NUOVO_LAVORATORE;
BEGIN
    select User, SysDate
        into User_Name, Entry_Date
        from DUAL;
    END REGISTRO_PACKAGE;
/

```

NOTA Il codice che deve essere eseguito a ogni esecuzione 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.

Ogniqualvolta il package REGISTRO_PACKAGE viene eseguito, le variabili User_Name ed Entry_Date vengono popolate per mezzo della query riportata nell'esempio precedente. Tali due variabili potranno quindi essere utilizzate dalle funzioni e procedure esistenti all'interno del package.

Per eseguire (execute) una procedura o 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 nell'esempio seguente:

```
execute REGISTRO_PACKAGE.CONTROLLO_SALDO('ADAH TALBOT');
```

24.8 Visualizzazione del codice sorgente di oggetti procedurali esistenti

Il codice sorgente di procedure, funzioni, package e corpi di package 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. |

Per selezionare informazioni dalla vista USER_SOURCE si utilizza una query simile a quella riportata nell'esempio seguente. In questo esempio è selezionata la colonna Text, ordinata per numero di riga. Il nome dell'oggetto (Name) e il suo tipo (Type) vengono utilizzati per definire gli oggetti il cui codice sorgente deve essere visualizzato. Nell'esempio riportato di seguito viene utilizzata la procedura NUOVO_LAVORATORE descritta in precedenza nel presente capitolo.

```
select Text
  from USER_SOURCE
 where Name = 'NUOVO_LAVORATORE'
   and Type = 'PROCÉDURE'
 order by Line;

TEXT
-----
procedure NUOVO_LAVORATORE
  (Nome_Persona IN varchar2)
AS
BEGIN
  insert into LAVORATORE
    (Nome, Eta, Alloggio)
  values
    (Nome_Persona, null, null);
END;
```

Come riportato nell'esempio precedente, la vista USER_SOURCE contiene un record per ciascuna riga della procedura NUOVO_LAVORATORE. La successione delle righe viene mantenuta dalla colonna Line. Per questo motivo la colonna Line dovrebbe essere utilizzata nella clausola order by, come nell'esempio.

Valori validi per la colonna Type sono PROCEDURE, FUNCTION, PACKAGE e PACKAGE BODY.

24.9 Compilazione di procedure, funzioni e package

ORACLE compila gli oggetti procedurali al momento della loro creazione, ma questi possono divenire non più validi se gli oggetti di database a cui fanno riferimento

cambiano. Alla successiva esecuzione degli oggetti procedurali, questi verranno ri-compilati dal database.

Per evitare questa compilazione durante l'esecuzione e il degrado delle prestazioni che essa può causare, è possibile ricompilare esplicitamente le procedure, le funzioni e i package. Per ricompilare una procedura si utilizza il comando alter procedure, come nell'esempio seguente. La clausola compile è l'unica opzione valida per tale comando.

```
alter procedure NUOVO_LAVORATORE compile;
```

Per utilizzare questo comando è necessario possedere la procedura o disporre del privilegio di sistema ALTER ANY PROCEDURE.

Per ricompilare una funzione si utilizza il comando alter function con la clausola compile.

```
alter function NUOVO_LAVORATORE compile;
```

Per utilizzare questo comando è necessario possedere la funzione o disporre del privilegio di sistema ALTER ANY PROCEDURE.

Per ricompilare i package è possibile sia ricompilare la specificazione e il corpo dello stesso, sia ricompilare il solo corpo. L'impostazione predefinita prevede la ricompilazione della specificazione e del corpo del package. 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 per le procedure o funzioni all'interno del corpo del package è cambiato, mentre la specificazione del package non è cambiata, è possibile ricompilare il solo corpo del package stesso. Nella maggior parte dei casi è più appropriato ricompilare sia la specificazione, sia il corpo del package.

Di seguito è riportata la sintassi per il comando alter package.

```
alter package [utente.] nome_package  
compile [PACKAGE | BODY];
```

Per ricompilare un package si utilizza il comando alter package descritto in precedenza con la clausola compile, come nell'esempio seguente.

```
alter package REGISTRO_PACKAGE compile;
```

Per utilizzare questo comando è necessario possedere il package 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', il cui risultato è la ricompilazione sia della specificazione, sia del corpo del package.

24.10 Sostituzione di procedure, funzioni e package

Le procedure, le funzioni e i package possono essere sostituiti mediante i rispettivi comandi create or replace. L'utilizzo della clausola or replace conserva invariato qualsiasi privilegio esistente creato per gli oggetti sostituiti. Se si eliminano e ricreano

oggetti procedurali, sarà necessario concedere nuovamente qualsiasi privilegio EXECUTE che fosse stato concesso in precedenza.

24.11 Scaricamento di procedure, funzioni e package

Per scaricare una procedura si utilizza il comando drop procedure, come nell'esempio seguente. È necessario possedere la procedura stessa o disporre del privilegio di sistema DROP ANY PROCEDURE.

```
drop procedure NUOVO_LAVORATORE;
```

Per scaricare una funzione si utilizza il comando drop function. È necessario possedere la funzione stessa oppure disporre del privilegio di sistema DROP ANY PROCEDURE.

```
drop function CONTROLLO_SALDO;
```

Per scaricare un package si utilizza il comando drop package, come nell'esempio seguente. È necessario possedere il package stesso o disporre del privilegio di sistema DROP ANY PROCEDURE.

```
drop package REGISTRO_PACKAGE;
```

Per eliminare il corpo di un package si utilizza il comando drop package con la clausola body, come nell'esempio seguente. È necessario possedere il package stesso o disporre del privilegio di sistema DROP ANY PROCEDURE.

```
drop package body REGISTRO_PACKAGE;
```

• Capitolo 25

• **Implementazione di tipi, viste oggetto e metodi**

- 25.1 **Nuova analisi dei tipi di dati astratti**
- 25.2 **Implementazione di viste oggetto**
- 25.3 **Metodi**

In questo capitolo sono riportate ulteriori informazioni sull'utilizzo di 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 inoltre descritta la creazione di metodi per i tipi di dati astratti, insieme all'utilizzo di un nuovo tipo di oggetto (la vista oggetto) e di un nuovo tipo di trigger (il trigger INSTEAD OF).

Per utilizzare le informazioni riportate nel presente capitolo è necessario conoscere i tipi di dati astratti (Capitolo 4), le viste (Capitolo 17) il comando grant (Capitolo 18), gli indici (Capitolo 19), le procedure e i trigger PL/SQL (Capitoli 22, 23 e 24). Ciascuno di questi argomenti gioca un ruolo sostanziale nell'implementazione del database relazionale a oggetti. Nel primo paragrafo del presente capitolo è riportata una seconda analisi dei tipi di dati astratti descritti nel Capitolo 4.

25.1 **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. Ad esempio, le colonne che fanno parte di informazioni relative a indirizzi possono essere raggruppate in un tipo di dati INDIRIZZO_TY per mezzo del comando create type.

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

Il comando create type dell'esempio precedente crea un tipo di dati astratto INDIRIZZO_TY, che può essere utilizzato per la creazione di ulteriori oggetti del database. Ad esempio, il comando create type riportato a pagina seguente crea il tipo di dati PERSONA_TY utilizzando il tipo di dati INDIRIZZO_TY come tipo di dati per la sua colonna Indirizzo.

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

La colonna Indirizzo del tipo di dati PERSONA_TY, dato che utilizza il tipo di dati INDIRIZZO_TY, non contiene un solo valore ma quattro: le quattro colonne correlate che costituiscono il tipo di dati INDIRIZZO_TY.

Sicurezza per i tipi di dati astratti

Nel precedente esempio si assumeva che lo stesso utente possedesse sia il tipo di dati INDIRIZZO_TY, sia il tipo di dati PERSONA_TY. Diverso sarebbe il caso in cui il proprietario del tipo di dati PERSONA_TY fosse differente dal proprietario del tipo di dati INDIRIZZO_TY.

Si supponga ad esempio che l'account “Dora” sia proprietario del tipo di dati INDIRIZZO_TY e che l'utente dell'account “George” cerchi 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 possiede il tipo di dati astratto INDIRIZZO_TY, ORACLE risponderà a tale comando create type con il messaggio riportato di seguito.

Warning: Type created with compilation errors.

Gli errori di compilazione sono causati da problemi nella creazione del *metodo costruttore*: 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 è proprietario di un tipo di dati con tale nome. George potrebbe impartire nuovamente il comando create type (utilizzando la clausola or replace) per fare specifico riferimento 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, occorre utilizzare il comando show errors (riportato in grassetto nell'esempio seguente).

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 è in grado di creare il tipo di dati PERSONA_TY (che comprende il tipo di dati INDIRIZZO_TY) se Dora non gli concede (grant) prima il privilegio EXECUTE sul suo proprio tipo di dati. Di seguito è riportata la concessione di tale privilegio.

```
grant EXECUTE on INDIRIZZO_TY to George;
```

Una volta ottenuti i privilegi richiesti, George sarà in grado di creare un tipo di dati basato sul tipo di dati INDIRIZZO_TY di Dora.

```
create or replace type PERSONA_TY as object
(Nome      VARCHAR2(25),
Indirizzo  Dora.INDIRIZZO_TY);
```

Il tipo di dati PERSONA_TY di George viene ora creato correttamente. L'utilizzo di tipi di dati basati su tipi di dati di un altro utente non è però banale. Ad esempio, durante le operazioni di insert è necessario specificare per esteso 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 comprende 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 insert in CLIENTE utilizzerebbe il formato riportato di seguito.

```
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 insert viene utilizzato il metodo costruttore INDIRIZZO_TY, posseduto da Dora. Pertanto il comando insert deve essere modificato in modo da specificare Dora come proprietario del tipo di dati INDIRIZZO_TY. Nell'esempio seguente è descritta la corretta istruzione insert, con il riferimento a Dora in grassetto.

```
insert into CLIENTE values
(1,PERSONA_TY('QualcheNome',
Dora.INDIRIZZO_TY('ValoreVia','ValoreCitta','ST',11111)))
```

George non è in grado di utilizzare un sinonimo per il tipo di dati di Dora, ma può invece creare un sinonimo denominato INDIRIZZO_TY.

```
create synonym INDIRIZZO_TY for Dora.INDIRIZZO_TY;
```

Tale sinonimo non può però essere utilizzato.

```
create type PERSONA2_TY
(Nome      VARCHAR2(25),
Indirizzo  INDIRIZZO_TY);
```

```
create type PERSONA2_TY
```

*

ERROR at line 1:
ORA-22863: synonym for datatype DORA.INDIRIZZO_TY not allowed

Come è riportato nel messaggio d'errore, non è possibile utilizzare un sinonimo per un tipo di dati di proprietà di un altro utente. Pertanto sarà necessario 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 cui il sinonimo viene creato. Se si impedisce 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 convalida dell'accessibilità di tali oggetti mediante sinonimi viene verificata solo quando si accede all'oggetto mediante il sinonimo.*

In un'implementazione esclusivamente relazionale di ORACLE, il privilegio EXECUTE viene concesso su oggetti procedurali, come procedure e package. All'interno 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. Per questo motivo il corretto privilegio da concedere è EXECUTE. Anche se Dora non definisce ancora alcun metodo sul tipo di dati INDIRIZZO_TY, ORACLE crea automaticamente procedure speciali, dette *metodi costruttori*, che vengono utilizzate per accedere ai dati. Qualsiasi oggetto (come PERSONA_TY) che utilizzi il tipo di dati INDIRIZZO_TY utilizzerà il metodo costruttore associato a INDIRIZZO_TY. Così, anche se non si è creato alcun metodo per tipo di dati astratto, esistono ancora procedure associate a quest'ultimo.

Indicizzazione degli attributi di un tipo di dati astratto

Nell'esempio del Capitolo 4 veniva creata la tabella CLIENTE. Come descritto nell'esempio seguente, tale tabella contiene una colonna normale, Cliente_ID, e una colonna Persona del tipo di dati astratto PERSONA_TY.

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

Dalle definizioni dei tipi di dati riportate nei paragrafi precedenti del presente capitolo si può evincere che PERSONA_TY dispone di una sola colonna, Nome, seguita da una colonna Indirizzo definita mediante il tipo di dati INDIRIZZO_TY.

Come riportato nel Capitolo 4, quando si fa riferimento a colonne all'interno dei tipi di dati astratti è necessario specificare il percorso completo sino agli attributi del tipo di dati. Ad esempio, la query riportata di seguito 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 ci si dovrà riferire all'attributo come a Persona.Nome:

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

Per fare riferimento ad attributi all'interno del tipo di dati INDIRIZZO_TY occorre specificare il percorso completo attraverso le colonne correlate. Ad 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 alla colonna Citta due volte: nell'elenco di colonne da selezionare e all'interno della clausola where.

```
select Persona.Nome,
       Person.Indirizzo.Citta
  from CLIENTE
 where Person.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 per la colonna Citta è disponibile un indice, ORACLE può individuare rapidamente tutte le righe i cui valori Citta iniziano con la lettera 'F', come definito nella query.

NOTA *Il funzionamento interno dell'ottimizzatore di ORACLE e le condizioni alle quali gli indici vengono utilizzati sono descritti nel Capitolo 36. Per consigli riguardanti le colonne da indicizzare si rimanda a tale capitolo.*

Per creare un indice o una colonna che faccia parte di un tipo di dati astratto è necessario 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);
```

Tale comando crea un indice di nome _CLIENTE\$CITTA sulla colonna Persona.Indirizzo.Citta. Ogni volta che accede alla colonna Citta, l'ottimizzatore di ORACLE valuta il codice SQL utilizzato per l'accesso ai dati e determina se il nuovo indice può essere utile per migliorare le prestazioni delle operazioni di accesso.

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, determinate colonne vengono normalmente utilizzate come parte di condizioni di limitazione all'interno di query, tali colonne devono essere indicizzate. A questo riguardo, la rappresentazione di più colonne in un singolo tipo di dati astratto può peggiorare le prestazioni dell'applicazione, in quanto può mascherare l'esigenza di indicizzare specifiche colonne all'interno del tipo di dati.

Utilizzando tipi di dati astratti, ci si abitua a trattare un gruppo di colonne come una singola 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 individualmente. Sarà pertanto necessario soddisfare i requisiti di indicizzazione per le colonne anche se si utilizzano tipi di

dati astratti. Inoltre, l'indicizzazione della colonna Citta in una tabella che utilizzi il tipo di dati INDIRIZZO_TY non influisce sulla colonna Citta presente in una seconda tabella che utilizzi il tipo di dati INDIRIZZO_TY. Se ad esempio esiste una seconda tabella di nome BRANCH 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. Il fatto che esista un indice per la colonna Citta nella tabella CLIENTE non ha nessun effetto sulla colonna Citta nella tabella BRANCH.

Pertanto, uno dei vantaggi dei tipi di dati astratti, la capacità di migliorare l'aderenza agli standard per la rappresentazione di dati logici, non si estende sino a comprendere la rappresentazione dei dati fisici. Ciascuna implementazione fisica deve essere curata separatamente.

25.2 Implementazione di viste oggetto

Per la tabella CLIENTE, creata nel precedente paragrafo, si assumeva che esistesse già un tipo di dati PERSONA_TY. Per l'implementazione di applicazioni che utilizzano database relazionali a oggetti è necessario utilizzare in primo luogo i metodi di progettazione per database relazionali descritti nella prima parte del volume. 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 nel presente capitolo.

Come ci si comporta se le tabelle esistono già? Come si procede se si è creata in precedenza un'applicazione di database relazionale e si desidera implementare i concetti relazionali a oggetti della propria applicazione senza ricostruire l'intera applicazione? In questo caso è necessario riuscire a sovrapporre strutture orientate agli oggetti (OO, Object-Oriented), come i tipi di dati astratti, su tabelle relazionali esistenti. ORACLE mette a disposizione delle *viste oggetto* per la definizione di oggetti utilizzati da tabelle relazionali esistenti.

NOTA *In tutti gli esempi del presente volume i nomi delle viste oggetto terminano sempre con il suffisso “OV”.*

Negli esempi riportati in precedenza nel presente capitolo, l'ordine delle operazioni era quello descritto di seguito:

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, è possibile creare i tipi di dati INDIRIZZO_TY e PERSONA_TY e utilizzare viste oggetto per correlare tali tipi di dati alla tabella CLIENTE. Nell'esempio seguente, la tabella CLIENTE viene creata come tabella relazionale, utilizzando solo i tipi di dati normalmente a disposizione.

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

Per creare un’ulteriore tabella o applicazione che memorizzi informazioni su persone e indirizzi, è possibile creare i tipi di dati INDIRIZZO_TY e PERSONA_TY. Per coerenza, questi dovrebbero essere applicati anche alla tabella CLIENTE.

Con la tabella CLIENTE già creata e possibilmente contenente dati, è possibile creare tipi di dati astratti. In primo luogo, si può creare INDIRIZZO_TY.

NOTA *Per gli esempi seguenti si assume che i tipi di dati INDIRIZZO_TY e PERSONA_TY non esistano già nel proprio schema.*

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

A questo punto occorre creare PERSONA_TY, che utilizza INDIRIZZO_TY:

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

Infine è possibile creare opzionalmente un oggetto basato su tutte le colonne della tabella CLIENTE. A tale oggetto viene assegnato il nome CLIENTE_TY.

```
create or replace type CLIENTE_TY as object
(CLiente_ID  NUMBER,
 Persona     PERSONA_TY);
```

Questo è un passaggio aggiuntivo, se confrontato al metodo descritto in precedenza nel presente paragrafo, che crea un terzo tipo di dati astratto CLIENTE_TY, contenente tutte le colonne all’interno della tabella CLIENTE, utilizzando nella relativa definizione il tipo di dati PERSONA_TY.

Si crei quindi 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 dell’oggetto CLIENTE_OV è riportato nell’esempio 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 questo comando. Nella prima riga, riportata di seguito, 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 definita mediante il tipo di dati PERSONA_TY). Non è possibile specificare “object” come opzione all'interno del comando `create view`.

Nella sezione successiva viene creata la query che formerà la base su cui poggia la vista.

```
select Cliente_ID,
       PERSONA_TY(Nome,
                  INDIRIZZO_TY(Via, Citta, Prov, Cap))
  from CLIENTE;
```

In questo esempio vi sono numerosi importanti problemi di sintassi. Quando una tabella viene costruita su tipi di dati astratti esistenti, i valori delle colonne vengono selezionati nella tabella facendo riferimento ai nomi delle colonne stesse (come Persona e Indirizzo) invece che ai loro metodi costruttori. Per la creazione di una vista oggetto è invece necessario fare comunque riferimento ai nomi dei metodi costruttori (PERSONA_TY e INDIRIZZO_TY).

NOTA *Per esempi di query che fanno riferimento a colonne definite mediante tipi di dati astratti si rimanda al Capitolo 4.*

Nelle query che formano la base della vista oggetto è possibile utilizzare clausole `where`. Nell'esempio seguente CLIENTE_OV viene modificato in modo da comprendere una clausola `where` che la limita alla sola visualizzazione dei valori CLIENTE per i quali la colonna Prov contenga il valore ‘DE’.

```
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 = 'DE';
```

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. Essa fa invece riferimento diretto alla tabella CLIENTE. La clausola `where` fa riferimento alla colonna Prov perché la query si riferisce direttamente alla tabella CLIENTE, una tabella relazionale che non è basata su tipi di dati astratti.

Per la creazione di viste oggetto basate su tabelle relazionali esistenti, si procede come segue.

1. Creare la tabella CLIENTE.
2. Creare il tipo di dati INDIRIZZO_TY.
3. Creare il tipo di dati PERSONA_TY, utilizzando il tipo di dati INDIRIZZO_TY.
4. Creare il tipo di dati CLIENTE_TY, utilizzando il tipo di dati PERSONA_TY.
5. Creare la vista oggetto CLIENTE_OV, utilizzando i tipi di dati definiti.

L'utilizzo di viste oggetto presenta due vantaggi principali. In primo luogo consente di creare tipi di dati astratti all'interno di tabelle preesistenti; dato che è possibile utilizzare gli stessi tipi di dati astratti in più tabelle all'interno dell'applicazione, è possibile anche migliorare l'aderenza dell'applicazione alla rappresentazione standard dei dati e la possibilità di riutilizzo di oggetti esistenti.

Inoltre, poiché è possibile definire metodi per i tipi di dati astratti, i metodi verranno applicati sia ai dati all'interno di nuove tabelle create, sia ai dati contenuti in tabelle preesistenti.

In secondo luogo, le viste oggetto consentono di utilizzare due differenti metodi per l'immissione dei dati nella tabella base, come viene spiegato nel paragrafo seguente. 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, è un vantaggio significativo per gli sviluppatori di applicazioni. Nel paragrafo seguente vengono descritte le opzioni di elaborazione dei dati disponibili per le viste oggetto.

Elaborazione di dati mediante viste oggetto

I dati nella tabella CLIENTE possono essere aggiornati per mezzo della vista oggetto CLIENTE_OV, oppure aggiornando direttamente la tabella CLIENTE. Trattando CLIENTE come una semplice tabella, è possibile effettuare inserimenti in essa mediante un normale comando SQL insert, come 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 dopo che è stata creata una vista oggetto sulla tabella, è sempre possibile trattare la tabella CLIENTE come una tabella relazionale convenzionale.

Dato che la vista oggetto è stata creata sulla tabella CLIENTE, è possibile effettuare il comando insert in CLIENTE attraverso i metodi costruttori utilizzati dalla vista. Come si è spiegato nel Capitolo 4, per effettuare inserimenti in un oggetto che utilizzi tipi di dati astratti è necessario specificare i nomi dei metodi costruttori all'interno del comando insert. Nell'esempio seguente è descritto l'inserimento di un singolo record nella vista oggetto CLIENTE_OV, utilizzando i metodi costruttori 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 ambedue i metodi possono essere utilizzati per effettuare operazioni di insert di valori nella vista oggetto CLIENTE_OV, è possibile standardizzare il modo in cui la propria applicazione effettua il trattamento dei dati. Se le operazioni di insert sono tutte basate su tipi di dati astratti, sarà possibile utilizzare lo stesso tipo di codice per le operazioni di insert, sia che i tipi di dati astratti siano stati creati prima, che dopo la tabella.

Utilizzo di trigger INSTEAD OF

Per la creazione di una vista oggetto è possibile utilizzare un trigger INSTEAD OF. Il trigger INSTEAD OF può essere utilizzato per una vista o per una tabella. Le viste non possono utilizzare le tabelle basate su trigger convenzionali, descritte nel Capitolo 23.

I trigger INSTEAD OF possono essere utilizzati per specificare a ORACLE come aggiornare le tabelle sottostanti che fanno parte della vista; si applicano sia a viste oggetto, sia a viste relazionali convenzionali.

Se ad esempio una vista comporta l'unione di due tabelle, la possibilità di aggiornare (update) i record nella vista è limitata. Utilizzando un trigger INSTEAD OF è possibile specificare a ORACLE come effettuare update, delete o insert di record nelle tabelle nel caso in cui l'utente cerchi di modificare valori tramite la vista. Il codice contenuto nel trigger INSTEAD OF viene eseguito al posto del comando insert, update o delete immesso.

Si considerino due delle più utilizzate tabelle di Talbot, LAVORATORE e ALLOGGIO. La tabella LAVORATORE prevede tre colonne. Nome, Eta e Alloggio. La tabella ALLOGGIO comprende quattro colonne: Alloggio, NomeLungo, Direttore e Indirizzo. È possibile selezionare tutte le colonne di ambedue le tabelle unendo queste ultime mediante la colonna Alloggio, come nell'esempio seguente:

```
select LAVORATORE.Nome,
       LAVORATORE.Eta,
       LAVORATORE.Alloggio,
       ALLOGGIO.Alloggio,
       ALLOGGIO.NomeLungo,
       ALLOGGIO.Direttore,
       ALLOGGIO.Indirizzo
  from LAVORATORE, ALLOGGIO
 where LAVORATORE.Alloggio = ALLOGGIO.Alloggio;
```

A questo punto è possibile selezionare da ciascuna tabella le colonne desiderate e utilizzare questa query come base per una vista. Ad esempio, è possibile selezionare dalle due tabelle il nome del lavoratore, il suo alloggio e il nome del Direttore dell'alloggio.

```
create view LAVORATORE_ALLOGGIO_DIRETTORE as
select LAVORATORE.Nome,
       ALLOGGIO.Alloggio,
       ALLOGGIO.Direttore
  from LAVORATORE, ALLOGGIO
 where LAVORATORE.Alloggio = ALLOGGIO.Alloggio;
```

È possibile selezionare valori da questa vista e, utilizzando il trigger INSTEAD OF, manipolare i dati mediante la vista stessa.

Si consideri il record riportato di seguito.

```
select *
  from LAVORATORE_ALLOGGIO_DIRETTORE
 where Nome = 'BART SARJEANT';
```

| NOME | ALLOGGIO | DIRETTORE |
|---------------|----------|--------------|
| BART SARJEANT | CRANMER | THOM CRANMER |

Bart Sarjeant risiede nell'alloggio 'CRANMER', gestito da Thom Cranmer. Egli ha però deciso di trasferirsi all'alloggio 'WEITBROCHT'. Ciò può essere indicato in due modi. È possibile effettuare direttamente l'aggiornamento della tabella LAVORATORE.

```
update LAVORATORE
  set Alloggio = 'WEITBROCHT'
  where Nome = 'BART SARJEANT';
```

È inoltre possibile utilizzare un trigger INSTEAD OF ed effettuare l'aggiornamento tramite la vista LAVORATORE_ALLOGGIO_DIRETTORE. Nell'esempio seguente è descritto un trigger INSTEAD OF per la vista LAVORATORE_ALLOGGIO_DIRETTORE.

```
create trigger LAVORATORE_ALLOGGIO_DIRETTORE_update
instead of UPDATE on LAVORATORE_ALLOGGIO_DIRETTORE
for each row
begin
  if :old.Nome <> :new.Nome
  then
    update LAVORATORE
      set Nome = :new.Nome
      where Nome = :old.Nome;
  end if;
  if :old.Alloggio <> :new.Alloggio
  then
    update LAVORATORE
      set Alloggio = :new.Alloggio
      where Nome = :old.Nome;
  end if;
  if :old.Direttore <> :new.Direttore
  then
    update ALLOGGIO
      set Direttore = :new.Direttore
      where Alloggio = :old.Alloggio;
  end if;
end;
```

Nella prima parte di questo trigger si assegna il nome a quest'ultimo e se ne descrive l'utilizzo per mezzo della clausola instead of. Il nome assegnato al trigger ne descrive il funzionamento: il trigger viene utilizzato per supportare comandi update eseguiti nei confronti della vista LAVORATORE_ALLOGGIO_DIRETTORE. Essendo questo un trigger a livello di riga, ciascuna riga modificata viene sottoposta a elaborazione.

```
create trigger LAVORATORE_ALLOGGIO_DIRETTORE_update
instead of UPDATE on LAVORATORE_ALLOGGIO_DIRETTORE
for each row
```

La sezione successiva del trigger specifica a ORACLE come elaborare il comando update. Durante il primo controllo all'interno del corpo del trigger viene verificato il valore Nome. Se il vecchio valore Nome è uguale al nuovo, non vengono effettuate modifiche. Se i valori non sono identici, la tabella LAVORATORE viene aggiornata con il nuovo valore Nome.

```
begin
  if :old.Nome <> :new.Nome
  then
    update LAVORATORE
      set Nome = :new.Nome
      where Nome = :old.Nome;
  end if;
```

Nella sezione successiva del trigger viene verificato il valore Alloggio. Se questo è cambiato, la tabella LAVORATORE viene aggiornata in modo da contenere il nuovo valore Alloggio per il lavoratore (LAVORATORE).

```
if :old.Alloggio <> :new.Alloggio
then
  update LAVORATORE
    set Alloggio = :new.Alloggio
    where Nome = :old.Nome;
end if;
```

Nella sezione finale del trigger viene verificata la colonna Direttore. Se questa è cambiata, la tabella ALLOGGIO viene aggiornata con il nuovo valore.

```
if :old.Direttore <> :new.Direttore
then
  update ALLOGGIO
    set Direttore = :new.Direttore
    where Alloggio = :old.Alloggio;
end if;
end;
```

In questo modo la vista è basata su due tabelle, LAVORATORE e ALLOGGIO, e un aggiornamento (update) della vista consente di aggiornare sia l'una, sia l'altra o ambedue le tabelle. Il trigger INSTEAD OF, ideato per supportare le viste oggetto, rappresenta un potente strumento per lo sviluppo di applicazioni.

È ora possibile effettuare l'operazione di update della vista LAVORATORE_ALLOGGIO_DIRETTORE direttamente, attraverso il trigger che effettua l'aggiornamento della tabella sottostante appropriata. Ad esempio, il comando riportato di seguito consente di aggiornare la tabella LAVORATORE.

```
update LAVORATORE_ALLOGGIO_DIRETTORE
  set Alloggio = 'WEITBROCHT'
  where Nome = 'BART SARJEANT';
```

```
1 row updated.
```

L'avvenuto update può essere verificato con una query della tabella LAVORATORE.

```
select Nome, Alloggio
  from LAVORATORE
 where Nome = 'BART SARJEANT';
```

| NOME | ALLOGGIO |
|---------------|------------|
| BART SARJEANT | WEITBROCHT |

È inoltre possibile aggiornare la tabella ALLOGGIO mediante la vista LAVORATORE_ALLOGGIO_DIRETTORE, nominando John Peletier nuovo gestore dell'alloggio di Bart Sarjeant.

```
update LAVORATORE_ALLOGGIO_DIRETTORE
  set Direttore = 'JOHN PELETIER'
 where Nome = 'BART SARJEANT';
```

1 row updated.

Per verificare la modifica, si selezionino le righe di John Peletier nella tabella ALLOGGIO.

```
select *
  from ALLOGGIO
 where Direttore = 'JOHN PELETIER';
```

| ALLOGGIO | NOMELUNGO |
|---------------|--------------------|
| DIRETTORE | INDIRIZZO |
| ROSE HILL | ROSE HILL FOR MEN |
| JOHN PELETIER | RFD 3, N. EDMESTON |
| WEITBROCHT | WEITBROCHT ROOMING |
| JOHN PELETIER | 320 GENEVA, KEENE |

I trigger INSTEAD OF sono molto potenti. Come illustrato in questo esempio, possono essere utilizzati per eseguire operazioni nei confronti di differenti tabelle del database, utilizzando la logica di controllo del flusso disponibile all'interno del linguaggio PL/SQL.

25.3 Metodi

Per la creazione di una vista oggetto sulla tabella CLIENTE si sono definiti tipi di dati astratti utilizzati dalla tabella stessa. Tali tipi di dati agevolano la standardizzazione della rappresentazione dei dati, ma hanno anche un altro scopo.

È possibile definire metodi che si applichino ai tipi di dati: applicando questi ultimi a tabelle esistenti sarà possibile applicare i metodi menzionati 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;
```

La vista oggetto CLIENTE_OV applica i tipi di dati astratti PERSONA_TY e INDIRIZZO_TY ai dati contenuti nella tabella CLIENTE. Se a tali tipi di dati sono associati 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 è necessario assegnare a quest'ultimo un nome all'interno della dichiarazione del tipo. Si crei ad esempio un nuovo tipo di dati, ANIMALE_TY. La definizione è riportata nell'esempio 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 GENERAZIONE utilizzata nel Capitolo 12. 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 riga:

```
member function ETA (DataNascita IN DATE) return NUMBER)
```

assegna un nome alla funzione che rappresenta un “membro” del tipo di dati ANIMALE_TY. Poiché “ETA”, restituisce un valore, si tratta di una funzione. Se non fosse restituito alcun valore, “ETA” sarebbe una procedura (Capitolo 24). Per definire la funzione ETA si utilizza il comando `create type body`, la cui sintassi è descritta nell'esempio seguente.

```
create [or replace] type body [utente.]tipo_Nome
  is
    member {dichiarazione_procedura | dichiarazione_funzione};
    [ member {dichiarazione_procedura | dichiarazione_funzione } ; ] ...
    [{map | order} member dichiarazione_funzione;]
  end;
```

Si crei la funzione ETA come funzione membro all'interno del tipo di dati ANIMALE_TY. Questa funzione ETA restituisce l'età, in giorni, degli animali.

```
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 ulteriori funzioni o procedure è necessario specificare le stesse all'interno dello stesso comando `create type body`, prima della clausola finale `end`.

Completata la creazione della funzione `ETA`, questa è associata con il tipo di dati `ANIMALE_TY`. Se una tabella utilizza tale 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`, come definita in precedenza nel comando `create type body`, non applica esplicite restrizioni agli aggiornamenti. Questa funzione non aggiorna alcun dato, ma non si hanno garanzie che il corpo del tipo non venga in seguito modificato per fare in modo che la funzione `ETA` aggiorni dei dati. Per richiamare una funzione membro all'interno di un'istruzione SQL è necessario assicurarsi che la funzione non sia in grado di aggiornare il database. Pertanto sarà necessario modificare la specifica della funzione nel comando `create type`. La modifica è riportata in grassetto nell'esempio 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 da una tabella, pertanto è assolutamente necessario effettuare questa modifica prima di creare una tabella che utilizzi il tipo di dati `ANIMALE_TY`.*

Si consideri la riga seguente.

```
PRAGMA RESTRICT_REFERENCES(ETA, WNDS);
```

Tale riga specifica a ORACLE che la funzione `ETA` opera nello stato Write No Database State e 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','01-JAN-97'));
```

È ora possibile richiamare la funzione ETA, che fa parte del tipo di dati ANIMALE_TY. Come accade con gli attributi dei tipi di dati astratti, ai metodi membro si fa riferimento mediante i nomi delle colonne che utilizzano i tipi di dati (in questo caso Animale.ETA).

```
select Animale.ETA(Animale.DataNascita)  
from ANIMALE;
```

```
ANIMALE.ETA(ANIMALE.DATANASCITA)
```

```
-----  
455
```

La chiamata Animale.ETA(Animale.DataNascita) esegue la funzione Eta all'interno del tipo di dati ANIMALE_TY. Pertanto non è necessario memorizzare dati variabili come le età all'interno del database; è invece possibile memorizzare dati statici come la data di nascita e utilizzare chiamate a funzioni e procedure per ricavare da essi i dati variabili.

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, è necessario modificarne il corpo, elencando tutti i metodi di quest'ultimo in un singolo comando.

Non è necessario concedere privilegi EXECUTE sulle funzioni membro e sulle procedure dei propri tipi di dati. Se si concede a un altro utente il privilegio EXECUTE sul tipo di dati ANIMALE_TY, tale utente disporrà automaticamente del privilegio EXECUTE sui metodi che fanno parte del tipo di dati. Così, 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 orientata agli oggetti.

Quando si creano funzioni membro è possibile definire le stesse come *metodi mappa* o *metodi ordine* (o non utilizzare nessuna delle due definizioni, come per la funzione membro ANIMALE_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 prende un record dell'oggetto come argomento esplicito e restituisce un valore intero. Se il valore restituito è negativo, zero o positivo, l'argomento "autonomo"隐式的 della funzione è rispettivamente minore, uguale o maggiore del valore del record esplicitamente specificato. Quando all'interno di un oggetto vengono confrontate più righe in una clausola order by, il metodo di ordinamento viene automaticamente eseguito per ordinare le righe restituite. Una specifica di tipo di dati può contenere solo un metodo ordine, che deve essere una funzione con tipo di restituzione INTEGER.

Il concetto relativo all'ordinamento all'interno delle righe di un tipo di dati astratto può avere maggiore rilevanza, 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, detti *tabelle annidate* e *array variabili*. Nel capitolo seguente è descritta l'implementazione di tali tipi di collettori.

• Capitolo 26

• **Tabelle annidate e array variabili**

• 26.1 **Array variabili**

• 26.2 **Tabelle annidate**

• 26.3 **Problemi nella gestione di tabelle annidate e array variabili**

ORACLE8 consente l'implementazione di caratteristiche relazionali in oggetti come i tipi di dati astratti e le viste oggetto. Nel presente capitolo viene descritto l'utilizzo dei *collektori*, serie di elementi trattati come parti di una singola riga. I due tipi di collektori disponibili sono le *tabelle annidate* e gli *array variabili*. Nei paragrafi seguenti vengono descritte le differenze tra i due tipi di collektori e la loro rispettiva implementazione e gestione.

I collektori utilizzano tipi di dati astratti. Per utilizzare gli array variabili e le tabelle annidate è consigliata una conoscenza approfondita della creazione e implementazione di questi tipi di dati (Capitolo 4 e Capitolo 25).

26.1 **Array variabili**

Un array variabile consente di memorizzare attributi ripetuti di un record in una singola riga. Ad esempio, si supponga che Dora Talbot desideri tenere traccia di quali attrezzi di sua proprietà siano stati presi in prestito dai suoi dipendenti. 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));
```

La chiave primaria della tabella PRESTITO è la combinazione della colonna Nome con la colonna Attrezzo. Così, se un singolo dipendente prendesse in prestito tre attrezzi, il suo nome verrebbe ripetuto in ciascuno dei tre record.

Il nome del dipendente, pur non cambiando, viene ripetuto in ciascun record in quanto fa parte della chiave primaria. I collektori come gli array variabili consentono di ripetere solo i valori della colonna che cambiano, risparmiando potenzialmente spazio su disco. I collektori possono essere utilizzati per rappresentare in modo accurato le relazioni tra tipi di dati negli oggetti del database. Nei paragrafi seguenti sono descritti la creazione e l'utilizzo dei collektori.

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). Ad esempio, è possibile creare un nuovo tipo di dati, ATTREZZO_TY. Tale tipo di dati prevede un'unica colonna. L'utilizzo di array variabili dovrebbe essere limitato 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 presente capitolo).

```
create type ATTREZZO_TY as object  
(NomeAttrezzo VARCHAR2(25));
```

Il tipo di dati astratto ATTREZZO_TY ha un attributo: NomeAttrezzo. Per utilizzare questo tipo di dati come parte di un array variabile in una tabella PRESTITO è necessario decidere quale sarà il massimo numero di attrezzi per dipendente. Per questo esempio si assume che non sia possibile prendere in prestito più di cinque attrezzi per persona.

Per creare l'array variabile (a cui viene 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 singolo passaggio. Per creare l'array variabile, si utilizzi la clausola `as varray()` del comando `create type`, come nell'esempio seguente.

```
create or replace type ATTREZZI_VA as varray(5) of VARCHAR2(25);
```

Quando il comando `create type` viene eseguito, si ha la creazione di un tipo ATTREZZI_VA. La clausola `as varray(5)` specifica 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 invece di ATTREZZO, per ricordare che questo tipo verrà utilizzato per contenere più record. Il suffisso "VA" rende inequivocabile il fatto che si tratta di un array variabile.

Una volta creato l'array variabile ATTREZZI_VA, è possibile utilizzarlo come parte della creazione di una tabella o di un tipo di dati astratto. Nell'esempio seguente viene creata la tabella PRESTITO utilizzando l'array variabile ATTREZZI_VA come tipo di dati per la sua colonna Attrezzo.

```
create table PRESTITO  
(Nome          VARCHAR2(25) primary key,  
 Attrezzi      ATTREZZI_VA);
```

Nella tabella PRESTITO, il nome della colonna che utilizza l'array variabile, Attrezzi, è al plurale per indicare che è in grado di contenere più valori per ciascun record PRESTITO. La colonna Nome è la chiave principale della tabella PRESTITO e non verrà pertanto ripetuta per ogni nuovo attrezzo preso in prestito.

Descrizione dell'array variabile

La tabella PRESTITO contiene un record per ciascuna persona che prende in prestito attrezzi, anche se questa ha in prestito più attrezzi. Tali attrezzi vengono memorizzati nella colonna Attrezzi, utilizzando l'array variabile ATTREZZI_VA.

Applicando il comando describe alla tabella PRESTITO è possibile vedere come ORACLE memorizzi internamente i dati di un array variabile utilizzando il tipo di dati RAW.

```
desc PRESTITO
```

| Name | Null? | Type |
|----------|----------|--------------|
| NOME | NOT NULL | VARCHAR2(25) |
| ATTREZZI | | RAW(200) |

Per visualizzare informazioni sulla struttura della colonna Attrezzi è possibile utilizzare la vista del dizionario di dati 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 |

Dal risultato di USER_TAB_COLUMNS si evince che la colonna Attrezzi utilizza l'array variabile ATTREZZI_VA come proprio tipo di dati. Per sapere di che genere di tipo di dati si tratta, è possibile effettuare una query sulla vista del dizionario di dati USER_TYPES.

```
select TypeCode,
       Attributes
  from USER_TYPES
 where Type_Name = 'ATTREZZI_VA';
```

| TYPECODE | ATTRIBUTES |
|------------|------------|
| COLLECTION | 0 |

Il risultato di USER_TYPE mostra che ATTREZZI_VA è un collettore, senza attributi.

Dato che ATTREZZI_VA è un collettore, potrà esistere un limite superiore al numero di voci per record che è in grado di contenere. Per visualizzare le caratteristiche dell'array variabile, compreso il tipo di dati astratto su cui si basa, è possibile effettuare una query sulla vista del dizionario di dati USER_COLL_TYPES.

```
select Coll_Type,
       Elem_Type_Owner,
       Elem_Type_Nome,
       Upper_Bound,
       Length
  from USER_COLL_TYPES
 where Type_Nome = 'ATTREZZI_VA'
```

| COLL_TYPE | ELEM_TYPE_OWNER | |
|----------------|-----------------|--------|
| ELEM_TYPE_NOME | UPPER_BOUND | LENGTH |
| VARYING ARRAY | | |
| VARCHAR2 | 5 | 25 |

NOTA Se si basa un array variabile su un tipo di dati astratto esistente, il proprietario del tipo di dati viene visualizzato nella colonna *Elem_Type_Owner* e il nome del tipo di dati viene visualizzato nella colonna *Elem_Type_Nome*. Dato che *ATTREZZI_VA* utilizza *VARCHAR2* invece di un tipo di dati astratto, il valore della colonna *Elem_Type_Owner* è *NULL*.

Dalle query sul dizionario di dati si ricava che la colonna PRESTITO.Attrezzi utilizza l'array variabile ATTREZZI_VA e che questo contiene un massimo di cinque valori la cui struttura viene definita come VARCHAR2(25). Tale informazione è necessaria per effettuare operazioni di insert di record nella tabella PRESTITO. Se l'array variabile si basa su un tipo di dati astratto, è possibile effettuare una query sulla vista dizionario dati USER_TYPE_ATTRS per determinare quali attributi sono parte del tipo di dati astratto. La struttura dei tipi di dati all'interno dell'array variabile determina la sintassi richiesta per l'inserimento di dati nella tabella PRESTITO.

Inserimento di record nell'array variabile

Quando viene creato un tipo di dati, il database crea automaticamente un metodo chiamato *metodo costruttore* per esso. Come descritto nel Capitolo 25, il metodo costruttore deve essere utilizzato per gli insert di 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 utilizzino array variabili è necessario impiegare i metodi costruttori. Inoltre, dato che l'array variabile è esso stesso un tipo di dati astratto, per effettuare l'operazione di insert di un record in una tabella che utilizza un array variabile può rendersi necessario annidare chiamate a più metodi costruttori.

Le colonne della tabella PRESTITO sono Nome e Attrezzi, la seconda delle quali è un array variabile che utilizza il tipo di dati ATTREZZI_VA. Il comando riportato di seguito effettua l'inserimento di un singolo record nella tabella PRESTITO. In questo esempio, il record presenta un singolo valore per la colonna Nome e tre valori per la colonna Attrezzi.

```
insert into PRESTITO values
('JED HOPKINS',
ATTREZZI_VA('HAMMER','SLEDGE','AX'));
```

Il precedente comando insert specifica in primo luogo il valore per la colonna Nome. Questa, non essendo parte di alcun tipo di dati astratto, viene riempita nello stesso modo in cui si effettuerebbero inserimenti di valori in una qualsiasi tabella relazionale.

```
insert into PRESTITO values
('JED HOPKINS',
```

La parte successiva del comando insert inserisce record nella colonna Attrezzi. Dato che questa utilizza l'array variabile ATTREZZI_VA, è necessario utilizzare il metodo costruttore ATTREZZI_VA.

```
ATTREZZI_VA('HAMMER','SLEDGE','AX'));
```

Poiché questo è un array variabile, è possibile passare più valori al metodo costruttore ATTREZZI_VA. Ciò è descritto nell'esempio seguente: per il singolo nome di un lavoratore vengono elencati tre attrezzi per un unico insert. Se l'array variabile fosse stato basato sul tipo di dati ATTREZZO_TY, sarebbe stato necessario annidare chiamate al tipo di dati ATTREZZO_TY all'interno dell'esecuzione di ATTREZZI_VA.

```
insert into PRESTITO values
('JED HOPKINS',
ATTREZZI_VA(
    ATTREZZO_TY('HAMMER'),
    ATTREZZO_TY('SLEDGE'),
    ATTREZZO_TY('AX')));
```

Un array variabile basato su un tipo di dati standard (come VARCHAR2 o NUMBER) invece che su un tipo di dati astratto semplifica pertanto il comando insert eliminando la necessità di annidare chiamate a metodi costruttori addizionali.

L'array variabile ATTREZZI_VA è stato definito in modo da prevedere sino a cinque valori.

Nel comando insert sono stati immessi solo tre valori nella riga, in tal modo gli altri valori rimangono non inizializzati. Per impostare i valori non inizializzati a NULL, è possibile specificare tale operazione nel comando insert.

```
insert into PRESTITO values
('JED HOPKINS',
ATTREZZI_VA('HAMMER','SLEDGE','AX',NULL,NULL));
```

Se si specificano troppi valori da inserire nell'array variabile, viene visualizzato il seguente messaggio d'errore:

ORA-22909: exceeded maximum VARRAY limit

Non è possibile effettuare insert di record in una tabella che contenga un array variabile se non si conosce la struttura dei tipi di dati all'interno dell'array. Nell'esempio precedente si assumeva che tutti i tipi di dati utilizzati dalla tabella PRESTITO fossero all'interno dello schema utilizzato per creare la tabella PRESTITO stessa.

Se i tipi di dati sono proprietà di uno schema differente, il creatore della tabella PRESTITO deve disporre del privilegio EXECUTE su tali tipi di dati. Durante le operazioni di insert, il creatore della tabella PRESTITO deve specificare esplicitamente il proprietario dei tipi di dati. Per un esempio sull'utilizzo di tipi di dati di proprietà di altri schemi si rimanda al Capitolo 25.

Selezione di dati dagli array variabili

Per effettuare insert di record in un array variabile è necessario assicurarsi che il numero di voci da inserire non superi il numero massimo consentito. Il numero massimo di voci in un array variabile viene specificato al momento della creazione dello stesso. Su tale numero può essere effettuata una query da USER_COLL_TYPES, come descritto nel paragrafo precedente del presente capitolo. È inoltre possibile effettuare una query diretta nell'array variabile per determinare il suo numero massimo di voci per riga, detto LIMIT dell'array variabile stesso, e il numero corrente di voci per riga, detto COUNT.

Non è però possibile effettuare una query diretta sull'array variabile per mezzo di un comando select. Per prelevare i valori COUNT, LIMIT e dati da un array variabile, è necessario utilizzare il linguaggio PL/SQL. Per effettuare una query sui dati da un array variabile, è possibile utilizzare una serie di cicli FOR a cursore anidati, come nell'esempio seguente.

NOTA *Per la descrizione del linguaggio PL/SQL e dei cicli FOR a cursore si rimanda al Capitolo 22.*

```
set serveroutput on
declare
  cursor Prestito_cursor is
    select * from PRESTITO;
  Prestito_rec  Prestito_cursor%ROWTYPE;
begin
  for Prestito_rec in Prestito_cursor
  loop
    dbms_output.put_line('Contact Nome: '||Prestito_rec.Nome);
    for i in 1..Prestito_rec.Attrezzi.Count
    loop
      dbms_output.put_line(Prestito_rec.ATTRIZZI(i));
    end loop;
  end loop;
end;
/
```

Il risultato di questo script PL/SQL è riportato di seguito.

```
Contact Nome: JED HOPKINS
HAMMER
SLEDGE
AX
```

PL/SQL procedure successfully completed.

Lo script utilizza più caratteristiche PL/SQL per prelevare i dati dall'array variabile. I dati vengono visualizzati mediante la procedura PUT_LINE del package DBMS_OUTPUT (Capitolo 24). Risulta quindi necessario in primo luogo abilitare la visualizzazione del risultato 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. Viene definita una variabile Prestito_rec con le stesse caratteristiche delle righe prelevate nel cursore.

```
declare
    cursor Prestito_cursor is
        select * from PRESTITO;
    Prestito_rec  Prestito_cursor%ROWTYPE;
```

La sezione dei comandi eseguibili del blocco PL/SQL contiene due cicli FOR annidati. Nel primo ciascun record del cursore viene valutato, mentre il valore di Nome viene stampato per mezzo della procedura PUT_LINE.

```
begin
    for Prestito_rec in Prestito_cursor
    loop
        dbms_output.put_line('Contact Nome: ' || Prestito_rec.Nome);
```

NOTA Per mezzo di PUT_LINE è possibile stampare solo dati letterali. Per questo motivo può rendersi necessario utilizzare la funzione TO_CHAR sui dati numerici prima di effettuare la stampa di questi ultimi mediante PUT_LINE.

Nella sezione successiva del blocco PL/SQL vengono ulteriormente valutate le righe, per recuperare i dati memorizzati nell'array variabile. Un secondo ciclo FOR a cursore, annidato all'interno del ciclo precedente, viene eseguito. Il ciclo FOR a cursore è limitato dal numero di valori nell'array, come definito dalla funzione COUNT (il numero massimo di valori nell'array può essere ottenuto per mezzo della funzione LIMIT). I valori presenti nell'array ATTREZZI vengono stampati, 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)

richiede a ORACLE la stampa del valore dell'array corrispondente al valore di *i*. Così, alla prima esecuzione del loop, viene stampato il primo valore nell'array. Il valore di *i* viene quindi incrementato e viene stampato 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;
/
```

La selezione di dati dagli array variabili non è banale, in quanto comporta cicli annidati all'interno di blocchi PL/SQL. Per una maggiore flessibilità nella selezione di dati dai collettori, è consigliato l'utilizzo di tabelle annidate, come descritto nel paragrafo successivo.

26.2 Tabelle annidate

Gli array variabili possono essere utilizzati per creare collettori all'interno di tabelle e tipi di dati. Mentre gli array variabili dispongono di 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 il suo nome suggerisce, una tabella all'interno di una tabella. In questo caso, è una tabella che è rappresentata da una colonna all'interno di un'altra tabella. Nella tabella annidata è possibile avere più righe per ciascuna riga della tabella principale.

Se ad esempio si ha una tabella di allevatori di animali, è possibile disporre anche di dati riguardanti gli animali dell'allevamento. Utilizzando una tabella annidata, è possibile 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 25.

```
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.

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 per una tabella annidata, è necessario creare un nuovo tipo di dati astratto.

```
create type ANIMALI_NT as table of ANIMALE_TY;
```

La clausola `as table of` del comando `create type` specifica a ORACLE che questo tipo di dati viene utilizzato come base per una tabella annidata. Il nome del tipo di dati, ANIMALI_NT, ha una radice al plurale per indicare che il tipo di dati memorizza più righe, e ha il suffisso 'NT' per indicare che si tratta di una tabella annidata.

È ora possibile creare una tabella di allevatori, utilizzando 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 è NomeAllevatore, la seconda 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 comprende una tabella annidata, è necessario specificare il nome della tabella utilizzata per memorizzare i dati della tabella annidata stessa. Ciò significa che i dati per la tabella annidata non sono memorizzati "in

“linea” con i restanti dati della tabella. Tali dati vengono invece memorizzati separatamente rispetto alla tabella principale. In questo modo, i dati nella colonna Animali vengono memorizzati in una tabella e i dati nella colonna Nome sono memorizzati in una tabella separata. ORACLE gestisce i puntatori tra le tabelle. In questo esempio, i dati “fuori linea” per la tabella annidata vengono memorizzati nella tabella ANIMALI_NT_TAB:

```
nested table ANIMALI store as ANIMALI_NT_TAB;
```

Questo esempio mette in evidenza l’importanza della standardizzazione dei nomi per i collezionatori. Se si utilizzano per le colonne nomi plurali (“Animali” invece di “Animale”) e si utilizzano coerentemente i suffissi (“TY,” “NT”), è possibile individuare la natura di un oggetto semplicemente a partire dal suo nome. Inoltre, dato che le tabelle annidate e gli array variabili possono essere basati 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 avrà una tabella associata fuori linea, denominata ANIMALI_NT_TAB. Incontrando una tabella di nome ANIMALI_NT_TAB, si potrebbe rilevare automaticamente che questa è basata su un tipo di dati di nome ANIMALE_TY.

Inserimento di record in una tabella annidata

Gli insert di record in una tabella annidata sono possibili utilizzando metodi costruttori per il tipo di dati della tabella. Per la colonna Animali, il tipo di dati è ANIMALI_NT; dovrà pertanto essere utilizzato il metodo costruttore ANIMALI_NT. Il tipo di dati ANIMALI_NT, a sua volta, utilizza il tipo di dati ANIMALE_TY. Come descritto nell’esempio seguente, l’inserimento di un record nella tabella ALLEVATORE richiede l’utilizzo di ambedue i metodi costruttori 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('DOG', 'BUTCH', '31-MAR-97'),
    ANIMALE_TY('DOG', 'ROVER', '05-JUN-97'),
    ANIMALE_TY('DOG', 'JULIO', '10-JUN-97')
));
```

Questo comando insert specifica, in primo luogo, il nome dell’allevatore.

```
insert into ALLEVATORE values
('JANE JAMES',
```

Una volta inserito il nome dell’allevatore, dovrà essere inserito il valore per la colonna Animali. Dato che tale colonna utilizza la tabella annidata ANIMALI_NT, nella riga seguente viene richiamato il metodo costruttore ANIMALI_NT.

```
ANIMALI_NT(
```

La tabella annidata ANIMALI_NT utilizza il tipo di dati ANIMALE_TY. Per ciascun record inserito viene pertanto richiamato il metodo costruttore ANIMALE_TY.

```
ANIMALE_TY('DOG', 'BUTCH', '31-MAR-97'),
ANIMALE_TY('DOG', 'ROVER', '05-JUN-97'),
ANIMALE_TY('DOG', 'JULIO', '10-JUN-97')
```

Le due parentesi finali completano il comando, chiudendo le chiamate al metodo costruttore ANIMALI_NT e l'elenco dei valori inseriti.

```
));
```

Se non si conosce già la struttura dei tipi di dati della tabella, è necessario effettuare una query sul dizionario di dati prima di essere in grado di effettuare query sulla tabella stessa. In primo luogo si esegua una query su USER_TAB_COLUMNS per visualizzare le definizioni delle colonne.

```
select Column_Name,
       Data_Type
  from USER_TAB_COLUMNS
 where Table_Name = 'ALLEVATORE';
```

| COLUMN_NAME | DATA_TYPE |
|----------------|------------|
| NOMEALLEVATORE | VARCHAR2 |
| ANIMALI | ANIMALI_NT |

Il risultato di USER_TAB_COLUMNS mostra che la colonna Animali utilizza il tipo di dati ANIMALI_NT. Per verificare che il tipo di dati ANIMALI_NT sia un collettore, è sufficiente controllare USER_TYPES:

```
select TypeCode,
       Attributes
  from USER_TYPES
 where Type_Name = 'ANIMALI_NT';
```

| TYPECODE | ATTRIBUTES |
|------------|------------|
| COLLECTION | 0 |

Per visualizzare il tipo di dati utilizzato come base per la tabella annidata è necessario eseguire una query su USER_COLL_TYPES:

```
select Coll_Type,
       Elem_Type_Owner,
       Elem_Type_Name,
       Upper_Bound,
       Length
  from USER_COLL_TYPES
 where Type_Name = 'ANIMALI_NT';
```

| COLL_TYPE | ELEM_TYPE_OWNER |
|-----------|-----------------|
|-----------|-----------------|

| ELEM_TYPE_NAME | UPPER_BOUND | LENGTH |
|----------------|-------------|--------|
| TABLE | TALBOT | |
| ANIMALE_TY | | |

Il risultato di USER_COLL_TYPES mostra che la tabella annidata ANIMALI_NT è basata sul tipo di dati astratto ANIMALE_TY. Per visualizzare gli attributi di ANIMALE_TY è possibile effettuare una query su USER_TYPE_ATTRS.

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

| ATTR_NAME | LENGTH | ATTR_TYPE_NAME |
|-------------|--------|----------------|
| GENERAZIONE | 25 | VARCHAR2 |
| NOME | 25 | VARCHAR2 |
| DATANASCITA | | DATE |

In questo modo è possibile utilizzare le viste del dizionario di dati per determinare la struttura dei tipi di dati delle tabelle annidate. Per utilizzare efficacemente la tabella ALLEVATORE è necessario conoscere le strutture dei tipi di dati e le strutture delle tabelle annidate. Nel paragrafo successivo viene descritta l'esecuzione di una query di dati da una tabella annidata. A differenza di quanto avviene per gli array variabili, le query su tabelle annidate non sono limitate dall'utilizzo del linguaggio PL/SQL. Per effettuare query su tabelle annidate è però richiesta una certa padronanza delle nuove estensioni SQL.

Query su tabelle annidate

A differenza degli array variabili, le tabelle annidate supportano una grande varietà di query. Durante le query stesse è però necessario tenere in considerazione la natura della tabella. Una tabella annidata è una colonna all'interno di una tabella. Per supportare query sulle colonne e alle righe di una tabella annidata, ORACLE mette a disposizione una nuova parola chiave: THE. Per comprendere come viene utilizzata questa parola chiave è necessario considerare in prima istanza la tabella annidata in sé. Se essa fosse una normale colonna di una tabella relazionale, sarebbe possibile effettuare una query per mezzo di un normale comando select.

```
select DataNascita /* Così non funziona.*/
   from ANIMALI_NT
  where Nome = 'JULIO';
```

Ma ANIMALI_NT non è una tabella normale: è un tipo di dati. Per selezionare colonne (come DataNascita) dalla tabella annidata, è necessario prima "appiattire" la tabella, in modo che sia possibile effettuare delle query su di essa. In questo caso si utilizza la funzione THE.

Si selezioni la colonna della tabella annidata dalla tabella principale, come nell'esempio seguente:

```
select Animali  
      from ALLEVATORE  
     where NomeAllevatore = 'JANE JAMES'
```

Quindi, si racchiuda questa query all'interno della funzione THE.

```
THE(select Animali  
      from ALLEVATORE  
     where NomeAllevatore = 'JANE JAMES')
```

È ora possibile effettuare query sulle colonne della tabella annidata utilizzando la clausola riportata nell'esempio precedente come *nome tabella* nella clausola from della query.

```
select NT.DataNascita  
      from THE(select Animali  
              from ALLEVATORE  
             where NomeAllevatore = 'JANE JAMES') NT  
       where NT.Nome = 'JULIO';
```

DATANASCITA

10-JUN-97

La funzione THE "appiattisce" la query della tabella annidata e consente di effettuare query sull'intera tabella ALLEVATORE. La clausola from contiene la query sulla colonna della tabella annidata e alla "tabella" risultante viene assegnato l'alias "NT".

```
from THE(select Animali  
          from ALLEVATORE  
         where NomeAllevatore = 'JANE JAMES') NT
```

La clausola where della query specifica le condizioni di limitazione per la query.

```
where NT.Nome = 'JULIO';
```

L'elenco delle colonne nella query specifica le colonne della tabella annidata da visualizzare.

```
select NT.DataNascita
```

In questo modo è possibile applicare criteri di limitazione sia ai valori all'interno della tabella annidata (il nome dell'animale) sia alla tabella principale (il nome dell'allevatore).

Ulteriori utilizzi della funzione THE La funzione THE può essere utilizzata ogniqualvolta sia necessario effettuare insert o update direttamente nella tabella annidata. Ad esempio, è probabile che un allevatore abbia con il passare del tempo sempre più animali. Sarà quindi necessario essere in grado di aggiungere nuovi record alla tabella annidata ANIMALI_NT all'interno della tabella ALLEVATORE.

Sarà necessario effettuare l'inserimento dei record nella tabella annidata senza aggiungere nuovi record alla tabella ALLEVATORE e riferire i nuovi record di animali agli allevatori già presenti nella tabella ALLEVATORE.

Si consideri l'istruzione insert riportata di seguito.

```
insert into
  THE(select Animali
       from ALLEVATORE
       where NomeAllevatore = 'JANE JAMES')
values
  (ANIMALE_TY('DOG', 'MARCUS', '01-AUG-97'));
```

In questo esempio, la funzione THE viene utilizzata al posto del nome di tabella. La clausola:

```
THE(select Animali
      from ALLEVATORE
      where NomeAllevatore = 'JANE JAMES')
```

seleziona all'interno della tabella principale ALLEVATORE la tabella annidata per la riga in cui la colonna NomeAllevatore ha il valore 'JANE JAMES'. È possibile effettuare le operazioni di insert direttamente nella tabella annidata utilizzando il metodo costruttore per il tipo di dati della tabella annidata stessa (ANIMALE_TY).

```
values
  (ANIMALE_TY('DOG', 'MARCUS', '01-AUG-97'));
```

Lo stesso metodo può essere utilizzato per effettuare update. Utilizzando la funzione THE al posto del nome tabella si è in grado di lavorare con i dati all'interno della tabella annidata.

Inserimenti basati su query Negli esempi di utilizzo della funzione THE, lo scopo delle query e delle operazioni DML è stato quello di manipolare i dati all'interno della tabella annidata. Per lavorare sulla tabella principale, è necessario nelle query un approccio leggermente differente.

Ad esempio, che cosa è necessario fare per eseguire un insert as select che coinvolga solo la porzione della tabella ALLEVATORE relativa alla tabella annidata? In pratica si desidera eseguire l'insert di un nuovo record nella tabella ALLEVATORE, ma i valori della tabella annidata risultano così basati su valori già inseriti per un altro record. L'allevatore Jane James ha deciso di coinvolgere nei propri affari un altro allevatore (Joan Thomas) e si desidera inserire nella tabella ALLEVATORE le informazioni relative a quest'ultimo.

È possibile cercare di effettuare semplicemente l'operazione di insert di valori nella tabella.

```
insert into ALLEVATORE values
  ('JOAN THOMAS'...)
```

Non è però possibile annidare direttamente i comandi select (per i valori della tabella annidata) all'interno di questo comando insert. È possibile cercare di annidare il nome del nuovo allevatore all'interno di una query sulla tabella annidata (utilizzando la funzione THE per effettuare una query sulla tabella annidata stessa).

```
insert into ALLEVATORE
select 'JOAN THOMAS', NT.Allevatore, NT.Nome, NT.DataNascita
from THE(select Animali
         from ALLEVATORE
        where NomeAllevatore = 'JANE JAMES') NT;
```

Anche questo tentativo non ha successo, in quanto i valori forniti sono troppi e la tabella ALLEVATORE contiene solo due colonne. Anche se una delle colonne è una tabella annidata, non è possibile effettuare insert diretti nelle sue colonne in questo modo.

Per risolvere questo problema, ORACLE dispone di due parole chiave: cast e multiset. La parola chiave cast consente di effettuare il *casting* del risultato di una query come tabella annidata. La parola chiave multiset consente alla query sottoposta a casting di contenere più record.

Le parole chiave cast e multiset vengono utilizzate insieme, come nell'esempio seguente, dove è riportato il record di ALLEVATORE sottoposto a insert, corretto in modo da utilizzare le parole chiave cast e multiset.

```
insert into ALLEVATORE values
('JOAN THOMAS',
 cast(multiset(
 select *
     from THE(select Animali from ALLEVATORE
              where NomeAllevatore = 'JANE JAMES'))
 as ANIMALI_NT));
```

Per controllare se il comando insert ha funzionato correttamente, è possibile effettuare una query sulla tabella ALLEVATORE per visualizzare gli animali contenuti nella tabella annidata del record 'JOAN THOMAS'.

```
select NT.Nome
  from THE(select Animali
           from ALLEVATORE
          where NomeAllevatore = 'JOAN THOMAS') NT;
```

| NOME |
|--------|
| BUTCH |
| ROVER |
| JULIO |
| MARCUS |

L'inserimento del record per Joan Thomas è stato completato con successo e tutte le voci della tabella annidata (colonna Animali) per Jane James sono state inserite anche come voci per Joan Thomas. Come ha funzionato il comando insert? In primo luogo il comando ha elencato il nuovo valore per la colonna NomeAllevatore, quello facente parte della tabella principale e non selezionato da una tabella annidata esistente.

```
insert into ALLEVATORE values
('JOAN THOMAS',
```

In seguito è stato necessario eseguire la sottoquery sulla tabella annidata. Le righe desiderate erano i valori della colonna Animali in cui l'allevatore era Jane James. Se la tabella annidata fosse stata una tabella relazionale (“piatta”), la query sarebbe stata quella riportata nell’esempio seguente.

```
select Animali from ALLEVATORE
  where NomeAllevatore = 'JANE JAMES';
```

Dato che Animali è una tabella annidata, è necessario utilizzare la funzione THE e select dalla tabella annidata come parte della clausola from di una query.

```
select *
  from THE(select Animali from ALLEVATORE
            where NomeAllevatore = 'JANE JAMES')
```

Poiché i valori della tabella annidata vengono utilizzati come parte di un’operazione di insert in una tabella principale, è necessario utilizzare le parole chiave cast e multiset per definire i risultati della query come tabella annidata.

```
cast(multiset(
    select *
      from THE(select Animali from ALLEVATORE
                where NomeAllevatore = 'JANE JAMES'))
    as ANIMALI_NT);
```

Si noti che il risultato della query sulla tabella annidata della colonna Animali è definito come “ANIMALI_NT”, il tipo di dati della tabella annidata utilizzato per definire la colonna Animali. Il comando combinato insert riportato di seguito è un esempio della potenzialità offerta dalle tabelle annidate. È possibile eseguire query e comandi di manipolazione dei dati direttamente sui valori della tabella annidata. In alternativa, essi possono essere utilizzati come parti di comandi che operano sulla tabelle principali. Per utilizzare efficacemente le tabelle annidate in SQL, è necessario conoscere l’utilizzo degli elementi THE, cast e multiset. Nel paragrafo successivo sono descritti ulteriori problemi di gestione correlati all’utilizzo di tabelle annidate e array variabili.

```
insert into ALLEVATORE values
('JOAN THOMAS',
 cast(multiset(
    select *
      from THE(select Animali from ALLEVATORE
                where NomeAllevatore = 'JANE JAMES'))
    as ANIMALI_NT));
```

26.3 Problemi nella gestione di tabelle annidate e array variabili

Esistono numerose differenze nei compiti amministrativi richiesti per le tabelle annidate e gli array variabili. Tali differenze sono descritte nei seguenti paragrafi, insieme ad alcuni suggerimenti su come individuare il tipo di collettore che meglio si adatta ai propri criteri operativi.

Gestione di grandi quantità di dati

Per la memorizzazione di dati correlati a una tabella è possibile scegliere un array variabile, una tabella annidata o una tabella separata. Se il numero delle righe è limitato, può essere più appropriato un array variabile. Via via che il numero di righe cresce, è possibile imbattersi in problemi di prestazioni durante l'accesso all'array variabile.

Tali problemi possono essere causati da una caratteristica degli array variabili: questi, infatti, non possono essere indicizzati. L'indicizzazione non è possibile nemmeno per le tabelle annidate, malgrado queste siano in grado di funzionare meglio degli array variabili nella gestione di più colonne. Le tabelle relazionali possono invece essere indicizzate. Per questi motivi, le prestazioni di un collettore possono peggiorare via via che le sue dimensioni aumentano.

Gli array variabili sono ulteriormente ostacolati dalla difficoltà che comporta l'effettuare query di dati su di essi (mediante blocchi PL/SQL invece di estensioni SQL). Se gli array variabili non sono una buona soluzione per un gran numero di record correlati, l'utilizzo di una tabella annidata o di una tabella relazionale separata dipende dagli obiettivi che si hanno. Le tabelle annidate e le tabelle relazionali servono a scopi differenti e 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, è possibile utilizzare tabelle annidate al posto di tabelle relazionali. In alternativa, è possibile utilizzare viste oggetto con metodi, come descritto nel Capitolo 25.
- Le tabelle relazionali possono essere facilmente correlate ad altre tabelle relazionali. Se i dati sono correlati a numerose altre tabelle, può essere consigliato non annidare i dati stessi all'interno di una tabella. La memorizzazione dei dati in una tabella propria mette a disposizione la maggiore flessibilità possibile nella gestione delle relazioni tra i dati medesimi.

Anche se il nome di una tabella annidata (come `ANIMALI_NT_TAB`) viene definito durante la creazione, non è possibile effettuare su di essa normali funzioni correlate alle tabelle. Ad esempio, non è possibile utilizzare il comando `describe` per la tabella.

```
desc ANIMALI_NT_TAB
```

ERROR:

```
ORA-22812: cannot reference nested table column's storage table
```

Quando per la memorizzazione dei dati si utilizza un collettore, i dati stessi non vengono memorizzati in una tabella che è possibile trattare come tabella relazionale. Per essere in grado di effettuare manipolazioni, è necessario creare la tabella come relazionale per mezzo del comando `create table` e non come tabella annidata all'interno di un'altra tabella.

Variabilità dei collettori

Come si è spiegato nel Capitolo 4, i tipi di dati astratti possono agevolare l'imposizione di standard all'interno del proprio database. Gli stessi vantaggi nella standardizzazione non possono però obbligatoriamente essere ottenuti quando si utilizzano i collettori. Si consideri l'array variabile ATTREZZI_VA descritto in precedenza nel presente capitolo. Se un'altra applicazione o tabella desidera utilizzare un array variabile ATTREZZI_VA, dovrà utilizzare per l'array la stessa definizione, compreso lo stesso numero massimo di valori. La nuova applicazione può però prevedere un valore differente per il massimo numero di attrezzi. Pertanto l'array deve essere modificato in modo da supportare il maggior numero 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 differenti limiti di array è necessario creare più array, rinunciando così al vantaggio nel riutilizzo degli oggetti consentito dalla creazione di array variabili. Sarà quindi necessario gestire i differenti limiti degli array e conoscere il limite di ogni singolo array.

Si può inoltre notare che più tabelle annidate sono simili, con solo una o due colonne differenti. Per questo motivo si può essere tentati di creare un singolo tipo di tabella annidata che comprenda tutto quanto e venga utilizzato in più tabelle. Anche tale tabella creerebbe però problemi di gestione.

Risulta infatti necessario sapere quali colonne della tabella annidata sono valide per le differenti implementazioni. Se non si è in grado di 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 tabelle annidate in tale ambiente, è necessario creare un tipo di dati di tabella annidata separato per ciascuna istanza, in cui si utilizza la tabella annidata stessa, e gestire tali 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 in una tabella con i rimanenti dati. Nelle tabelle annidate, i dati possono essere memorizzati fuori linea. In questo modo il numero di dati analizzati durante le query che non comprendano i dati del collettore, può essere minore per una tabella annidata dello stesso numero per un array variabile. Durante una query il database non avrà bisogno di leggere tutti i dati nella tabella annidata; se viene utilizzato un array variabile, il database potrà limitarsi a leggere i dati dell'array variabile stesso, per individuare i dati cercati. I dati fuori linea, come utilizzati dalle tabelle annidate, possono migliorare le prestazioni delle query.

I dati fuori linea dei collettori emulano il modo in cui i dati verrebbero memorizzati in una normale applicazione di database relazionale. Ad 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, ma conservano una relazione con questa. I dati fuori linea possono migliorare le prestazioni delle query distribuendo gli accessi I/O alla tabella su più tabelle separate.

I dati fuori linea vengono inoltre utilizzati per memorizzare oggetti LOB (Large OBjects) che vengono memorizzati internamente. Nel capitolo seguente viene descritta la creazione e la manipolazione di valori LOB sino a 4 GB di dimensione. La combinazione di tipi di dati astratti, viste oggetto, collezionatori e LOB mette a disposizione solide fondamenta per l'implementazione di un'applicazione di database relazionale a oggetti. Nel Capitolo 31 viene descritto l'ulteriore ampliamento di tali oggetti, per andare verso la creazione di un database orientato all'oggetto.

• Capitolo 27

• Utilizzo di LOB in ORACLE8

• 27.1 Tipi di dati disponibili

• 27.2 Definizione dei parametri di memorizzazione per i dati LOB

• 27.3 Gestione e selezione dei valori LOB

Per memorizzare dati alfanumerici sino a 2 GB di lunghezza per riga è possibile utilizzare un tipo di dati LONG. Il tipo di dati LONG RAW consente la memorizzazione di dati binari di lunghezza considerevole. In ORACLE8 sono disponibili nuovi tipi di dati per la memorizzazione di dati di notevole lunghezza. Utilizzando uno dei nuovi tipi di dati per memorizzare grandi oggetti (LOB) è possibile trarre vantaggio dalle nuove possibilità di visualizzazione e manipolazione dei dati. Nel presente capitolo viene descritto l'utilizzo dei nuovi tipi di dati e la manipolazione di dati di notevole lunghezza, anche se non memorizzati all'interno del database.

27.1 Tipi di dati disponibili

In ORACLE8.0 sono disponibili quattro tipi di oggetti LOB. I tipi di dati e le relative descrizioni sono riportati nella Tabella 27.1.

Tabella 27.1 Tipi di dati LOB.

| TIPO DI DATI LOB | DESCRIZIONE |
|------------------|---|
| BLOB | LOB binario, dati binari, sino a 4 GB di lunghezza, memorizzati nel database. |
| CLOB | LOB alfanumerico, dati alfanumerici; sino a 4 GB di lunghezza, memorizzati nel database. |
| BFILE | File binario, dati binari a sola lettura memorizzati all'esterno del database; la lunghezza è limitata dal sistema operativo. |
| NCLOB | Colonna CLOB che supporta un set di caratteri multibyte. |

Per ogni tabella è possibile creare più LOB. Ad esempio, Dora Talbot è in grado di creare una tabella PROPOSTA per tenere traccia delle proposte formali presentate. I record relativi alle proposte di Dora consistono in una serie di file creati con elaboratori di testi e fogli elettronici, utilizzati per documentare i prezzi e i lavori

proposti. 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 elettronico). Nell'esempio seguente, il comando `create table` consente di creare la tabella PROPOSTA.

```
create table PROPOSTA
(Proposta_ID      NUMBER(10) primary key,
Nome_Destin      VARCHAR2(25),
Proposta_Nome    VARCHAR2(25),
Breve_Desrizione VARCHAR2(1000),
Proposta_Testo   CLOB,
Budget           BLOB,
Lettera_Present  BFILE);
```

Poiché Dora può sottoporre più proposte a un singolo destinatario, a ciascuna proposta viene assegnato un numero `Proposta_ID`. Per ciascuna proposta vengono memorizzati il destinatario, il nome e una breve descrizione. Quindi, avvantaggiandosi dei nuovi tipi di dati LOB disponibili in ORACLE, è possibile aggiungere tre ulteriori colonne.

| | |
|-----------------|--|
| Proposta_Testo | CLOB contenente il testo della proposta. |
| Budget | BLOB contenente un foglio elettronico che descrive i calcoli relativi ai costi e ai ricavi per il lavoro proposto. |
| Lettera_Present | File binario memorizzato all'esterno del database, che contiene la lettera d'accompagnamento di Dora allegata alla proposta. |

In questo modo, per ciascuna riga della tabella PROPOSTA vengono memorizzati sino a tre valori LOB. ORACLE utilizza le normali possibilità del database nel supporto dell'integrità dei dati e della corrispondenza tra le voci `Proposta` e `Budget`, ma non nel supporto dei 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 stesso memorizza solo un valore di locazione che consente di individuare il file esterno. ORACLE non garantisce l'integrità dei file BFILE memorizzati all'esterno del database. Inoltre ORACLE non convalida l'esistenza del file quando si inserisce un record utilizzando un tipo di dati BFILE. La corrispondenza e l'integrità dei dati sono preservate solamente per i LOB memorizzati internamente.

I dati per le colonne LOB, siano essi memorizzati all'interno o all'esterno del database, non sempre vengono memorizzati fisicamente assieme alla tabella PROPOSTA. All'interno della tabella PROPOSTA, ORACLE memorizza valori che puntano alle locazioni dei dati. Per i tipi di dati BFILE, il puntatore di localizzazione 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. In questo modo i dati LOB non vengono obbligatoriamente memorizzati direttamente assieme 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 localizzazione. È possibile specificare parametri di memorizzazione (tablespace e dimensionamento) per l'area utilizzata per conservare i dati LOB, come descritto nel paragrafo seguente.

27.2 Definizione dei parametri di memorizzazione per i dati LOB

Quando si crea una tabella contenente una colonna LOB, è possibile specificare i parametri di memorizzazione relativi all'area utilizzata per i dati LOB. In questo modo, in una tabella che non contenga colonne LOB, è possibile inserire una clausola storage nel comando create table (Capitolo 19). Se la tabella contiene almeno una colonna LOB risulta necessaria un'ulteriore clausola lob all'interno del comando create table.

Si consideri la tabella PROPOSTA di Dora, questa volta con clausole storage e tablespace.

```
create table PROPOSTA
(Proposta_ID      NUMBER(10) primary key,
 Nome_Destin      VARCHAR2(25),
 Proposta_Nome    VARCHAR2(25),
 Breve_Descrizione VARCHAR2(1000),
 Proposta_Testo   CLOB,
 Budget           BLOB,
 Lettera_Present  BFILE)
storage (initial 50K next 50K pctincrease 0)
tablespace PROPOSTE;
```

Poiché sono presenti tre colonne LOB, il comando create table deve essere modificato in modo da comprendere le specifiche di memorizzazione per i dati LOB fuori linea. Due delle colonne LOB, Proposta_Testo e Budget, registrano i dati all'interno del database. La versione corretta del comando create table è riportata nell'esempio seguente:

```
create table PROPOSTA
(Proposta_ID      NUMBER(10) primary key,
 Nome_Destin      VARCHAR2(25),
 Proposta_Nome    VARCHAR2(25),
 Breve_Descrizione VARCHAR2(1000),
 Proposta_Testo   CLOB,
 Budget           BLOB,
 Lettera_Present  BFILE)
storage (initial 50K next 50K pctincrease 0)
tablespace PROPOSTE
lob (Proposta_Testo, Budget) store as
(tablespace Proposta_Lobs
storage (initial 100K next 100K pctincrease 0)
chunk 16K pctversion 10 nocache logging);
```

Le ultime quattro righe del comando `create table` specificano a ORACLE in che modo memorizzare i dati LOB fuori linea. Di seguito, sono riportate tali istruzioni:

```
lob (Proposta_Testo, 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 la successiva serie di comandi si riferisce alle specifiche di memorizzazione fuori linea per le colonne LOB della tabella. Le due colonne LOB (Proposta_Testo e Budget) vengono elencate esplicitamente. I valori delle due colonne, se memorizzati fuori linea, vengono registrati in un segmento, come specificato mediante la clausola `lob`.

```
lob (Proposta_Testo, Budget) store as
```

NOTA *Se la colonna LOB fosse una sola, sarebbe possibile specificare un nome per il segmento LOB. Il nome del segmento viene riportato immediatamente dopo la clausola store as nel comando specificato in precedenza.*

Le due righe successive specificano i parametri relativi al tablespace e alla memorizzazione fuori linea per i LOB. La clausola `tablespace` assegna i dati fuori linea al tablespace PROPOSTA_LOBS. La clausola `storage` viene utilizzata per specificare i valori `initial`, `next` e `pctincrease` utilizzati per la memorizzazione fuori linea dei dati. Per ulteriori informazioni sui parametri di memorizzazione disponibili per i segmenti, si rimanda al paragrafo dedicato alla memorizzazione nella Guida alfabetica di riferimento del Capitolo 37.

```
(tablespace Proposta_Lobs
  storage (initial 100K next 100K pctincrease 0))
```

Poiché i segmenti vengono utilizzati per la memorizzazione di dati LOB, sono disponibili numerosi ulteriori parametri, specificati all'interno della clausola `lob`.

```
chunk 16K pctversion 10
```

Il parametro di memorizzazione di LOB `chunk` specifica a ORACLE quanto spazio allocare durante ciascuna manipolazione di valori LOB. Le dimensioni predefinite per `chunk` sono pari a 1 K e possono arrivare a un massimo di 32 K.

Il parametro `pctversion` rappresenta la percentuale massima di spazio complessivo di memorizzazione di LOB utilizzato per la creazione di nuove versioni del LOB. Per default le versioni precedenti dei dati LOB non vengono riscritte sino a che non viene utilizzato il 10 per cento dello spazio di memorizzazione di LOB disponibile.

Gli ultimi due parametri di memorizzazione di LOB specificano a ORACLE come gestire i dati LOB durante la lettura e la scrittura.

```
nocache logging);
```

Il parametro `nocache` nella clausola di memorizzazione `lob` specifica che i valori LOB non vengono memorizzati in memoria per un più veloce accesso durante le query. L'impostazione predefinita di memorizzazione `lob` è `nocache`. Il suo contrario è `cache`.

Il parametro logging specifica che tutte le operazioni nei confronti dei dati LOB vengono registrate nei file di redo log del database. Al contrario di logging, nologging non viene effettuata registrazione delle operazioni nei file di redo log, migliorando le prestazioni durante le operazioni di creazione delle tabelle. Il parametro nologging disponibile in ORACLE8.0 prende il posto del parametro unrecoverable disponibile in ORACLE7.2.

È inoltre possibile specificare i parametri tablespace e storage per un indice LOB. Per la sintassi completa relativa agli indici LOB, si consulti la voce relativa al comando create table nella Guida alfabetica di riferimento del Capitolo 37. Una volta creata la tabella, Dora è in grado di iniziare l'inserimento dei record in questa. Nel paragrafo successivo, viene descritto l'inserimento dei valori LOB nella tabella PROPOSTA e l'utilizzo del package DBMS_LOB per la gestione dei valori.

27.3 Gestione e selezione dei valori LOB

I dati LOB possono essere selezionati o manipolati in molti modi. Il metodo di manipolazione dei dati LOB più comune prevede l'utilizzo del package DBMS_LOB. Altri metodi per la manipolazione dei dati comprendono l'utilizzo di API (Application Programming Interfaces) e OCI (ORACLE Call Interface). Nel presente paragrafo è descritto l'utilizzo del package DBMS_LOB per la manipolazione dei dati LOB. Come si può vedere dagli esempi seguenti, i LOB sono molto più flessibili dei tipi di dati LONG in termini di possibilità di manipolazione dei dati. Essi non sono però così semplici da selezionare e manipolare quanto le colonne VARCHAR2.

Inizializzazione dei valori

Si consideri ancora la tabella PROPOSTA di Dora, visualizzata nella Tabella 27.2. La colonna Proposta_ID è la colonna chiave primaria della tabella PROPOSTA. Nella tabella PROPOSTA sono presenti tre colonne LOB: una colonna BLOB, una colonna CLOB e una colonna BFILE. Per ciascuna delle colonne LOB, ORACLE memorizza un *locatore* che indica la locazione dei dati fuori linea memorizzati per il record.

Tabella 27.2 Colonne della tabella PROPOSTA.

| NOME COLONNA | TIPO DI DATI |
|-------------------|----------------|
| Proposta_ID | NUMBER(10) |
| Nome_Destin | VARCHAR2(25) |
| Proposta_Nome | VARCHAR2(25) |
| Breve_Descrizione | VARCHAR2(1000) |
| Proposta_Testo | CLOB |
| Budget | BLOB |
| Lettera_Present | BFILE |

Quando si inserisce un record in una tabella che contiene LOB, si utilizza il package DBMS_LOB per fare in modo che ORACLE crei un locatore vuoto per le colonne LOB memorizzate internamente. Un locatore vuoto è differente da un locatore con valore NULL. Se il valore di una colonna LOB memorizzata internamente è NULL, è necessario trasformarlo in un valore di locatore vuoto prima di aggiornarlo a un valore non-NUL.

Si supponga di iniziare a lavorare su una proposta e di inserire un record nella tabella PROPOSTA. A questo punto, non si dispone di un foglio elettronico Budget né di una lettera d'accompagnamento. Il comando insert che si può utilizzare è riportato nell'esempio seguente.

```
insert into PROPOSTA
(Proposta_ID, Nome_Destin, Proposta_Nome, Breve_Descrizione,
Proposta_Testo,
Budget, Lettera_Present)
values
(1, 'DOT PHILLIPS', 'AZZERAMENTO CAMPO PHILLIPS', NULL,
'Questo è il testo di una proposta di azzeramento del campo di PHILLIPS'
EMPTY_BLOB(),NULL);
```

Il record inserito ha un Proposta_ID pari 1, e un Nome_Destin ‘DOT PHILLIPS’. Il valore di Proposta_Nome è ‘AZZERAMENTO CAMPO PHILLIPS’ e la colonna Breve_Descrizione è per ora lasciata a NULL. La colonna Proposta_Testo è per il momento uguale a una breve stringa di caratteri: ‘Questo è il testo di una proposta di azzeramento del campo di PHILLIPS’. Per assegnare alla colonna Budget un locatore vuoto, viene utilizzata la funzione EMPTY_BLOB. Per impostare una colonna di tipo CLOB uguale al valore di un locatore vuoto, si è utilizzata la funzione EMPTY_CLOB. La colonna Lettera_Present, essendo un valore BFILE memorizzato esternamente, viene impostata a NULL.

Per impostare in una colonna LOB memorizzata internamente un locatore vuoto, è necessario conoscere il tipo di dati della colonna.

| | |
|-------|-------------------------|
| BLOB | Utilizzare EMPTY_BLOB() |
| CLOB | Utilizzare EMPTY_CLOB() |
| NCLOB | Utilizzare EMPTY_CLOB() |

La procedura BFILENAME consente di puntare a directory e file. Prima di specificare un valore per la directory, è necessario che un utente con il ruolo DBA o il privilegio di sistema CREATE DIRECTORY crei la directory stessa. Per creare una directory si utilizza il comando create directory, come nell'esempio seguente.

```
create directory 'dir_proposta' for '/U01/PROPOSTA/LETTERS';
```

Quando si inseriscono voci BFILE, è necessario fare riferimento al nome della directory logica, come ‘dir_proposta’, anziché 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 della directory. Per ulteriori informazioni, si faccia riferimento al paragrafo relativo al comando create directory nella Guida alfabetica di riferimento.

È ora possibile inserire un secondo record PROPOSTA con un valore di Lettera_Present.

```
insert into PROPOSTA
(Proposta_ID, Nome_Destin, Proposta_Nome, Breve_Descrizione,
Proposta_Testo, Budget,
Lettera_Present)
values
(2, 'BRAD OHMONT', 'RICOSTRUIRE RECINZIONE', NULL,
EMPTY_CLOB(), EMPTY_BLOB(),
BFILENAME('dir_proposta','P2.DOC'));
```

In questo esempio, nella tabella PROPOSTA viene inserita una seconda proposta: ricostruire una recinzione. Le funzioni EMPTY_CLOB() ed EMPTY_BLOB() vengono utilizzate per inserire locatori vuoti nella tabella. La funzione BFILENAME specifica a ORACLE dove esattamente trovare la lettera d'accompagnamento, nella directory ‘dir_proposta’ e con il nome P2.DOC. All’interno di BFILENAME il primo parametro è sempre il nome della directory e il secondo parametro è il nome del file all’interno della directory stessa.

Quando si seleziona il valore di una colonna LOB, ORACLE utilizza il valore del locatore per individuare i dati associati con i dati LOB. Non è mai necessario conoscere o specificare i valori dei locatori. Inoltre, come viene spiegato nei paragrafi successivi, i valori LOB consentono operazioni impossibili con i tipi di dati LONG.

Inserimento con sottoquery

A questo punto, viene descritta la duplicazione del record di una proposta. Si supponga di iniziare a lavorare su una terza proposta, molto simile alla prima.

```
insert into PROPOSTA
(Proposta_ID, Nome_Destin, Proposta_Nome, Breve_Descrizione,
Proposta_Testo, Budget, Lettera_Present)
select 3, 'SALTA GATES', 'AZZERAMENTO CAMPO GATES', NULL,
Proposta_Testo, Budget, Lettera_Present
from PROPOSTA
where Proposta_ID = 1;
```

Il comando insert ordina a ORACLE di trovare il record PROPOSTA con valore Proposta_ID pari a 1. Sarà quindi necessario prendere i valori delle colonne Proposta_Testo, Budget e Lettera_Present e i valori letterali specificati (3, ‘SALTA GATES’ e così via) e inserire un nuovo record nella tabella PROPOSTA. Si noti che le colonne selezionate comprendono colonne LOB. Se le colonne LOB contengono locatori vuoti, le colonne inserite vengono impostate su valori di locatori vuoti. La possibilità di effettuare insert basandosi su query rappresenta un significativo vantaggio per quanto riguarda l’utilizzo dei tipi di dati LOB. Questa modalità di insert non può essere utilizzata se una colonna LONG fa parte della query.

NOTA *Se si utilizza insert as select per inserire i valori LOB, è possibile che più valori BFILE puntino allo stesso file esterno. Potrà rendersi necessaria un’operazione di update dei nuovi valori in modo che questi puntino ai file esterni corretti. ORACLE non conserva l’integrità dei dati per i file esterni.*

Aggiornamento dei valori LOB

Prima di modificare un valore LOB è necessario bloccare (con un “lock”) la riga da aggiornare. Normalmente il blocco delle righe avviene in background: si esegue un comando update e ORACLE blocca dinamicamente la riga, effettua la transazione e quindi sblocca la riga. Se si aggiorna una riga che contiene un valore LOB, è necessario bloccare esplicitamente la riga prima di eseguire il comando update. Altrimenti il comando non verrà completato con successo.

Nel terzo record creato, il valore Proposta_Testo è stato copiato dal primo record. Se si desidera aggiornare il valore Proposta_Testo del record che ha Proposta_ID pari a 3, occorre innanzitutto bloccare il record. Per bloccare il record, si seziona il record stesso utilizzando la clausola for update.

```
select Proposta_Testo
  from PROPOSTA
  where Proposta_ID = 3
    for update;
```

Una volta selezionato il record di Proposta_ID pari a 3 e specificata la clausola for update, si mantiene un blocco esclusivo sulla riga. È ora possibile aggiornare il contenuto della riga con il nuovo valore di Proposta_Testo.

```
update PROPOSTA
  set Proposta_Testo = 'Questo è il nuovo testo della proposta.'
  where Proposta_ID = 3;
```

Per rimuovere il blocco, si può applicare commit alla modifica.

```
commit;
```

È inoltre possibile eseguire l'operazione di update della colonna Lettera_Present in modo da puntare alla corretta lettera d'accompagnamento. Per puntare al file corretto, è possibile utilizzare la funzione BFILENAME.

```
update PROPOSTA
  set Lettera_Present = BFILENAME('dir_proposta', 'P3.DOC')
  where Proposta_ID = 3;
```

Dato che la colonna Lettera_Present utilizza il tipo di dati BFILE, non è necessario bloccare, con la clausola for update, la riga prima di effettuare l'aggiornamento: i blocchi esclusivi sono necessari solo per aggiornamenti di LOB memorizzati internamente. Inoltre è possibile aggiornare i valori NULL delle colonne BFILE senza dover prima impostare locatori 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 locatore vuoto, è possibile effettuare l'aggiornamento del valore della colonna Budget.

Utilizzo 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 viene descritta l'esecuzione di funzioni di SQL come SUBSTR e INSTR rispetto a valori LOB, l'aggiunta a valori LOB, il confronto e la lettura di tali valori.

Le procedure di confronto e manipolazione di stringhe disponibili nel package DBMS_LOB sono elencate nella Tabella 27.3. Compaiono per prime le procedure e funzioni che selezionano i dati, seguite dalle procedure e dalle funzioni che modificano i dati.

Tabella 27.3 Procedure e funzioni presenti nel package DBMS_LOB.

| PROCEDURA O FUNZIONE | DESCRIZIONE |
|----------------------|--|
| READ | Procedura utilizzata per leggere una parte di un valore LOB. |
| SUBSTR | Funzione utilizzata per eseguire la funzione SQL SUBSTR su un valore LOB. |
| INSTR | Funzione utilizzata per eseguire la funzione SQL INSTR su un valore LOB. |
| GETLENGTH | Funzione utilizzata per eseguire la funzione SQL LENGTH su un valore LOB. |
| COMPARE | Funzione utilizzata per confrontare due valori LOB. |
| WRITE | Procedura utilizzata per scrivere dati nel valore LOB in un punto determinato del valore LOB stesso. |
| APPEND | Procedura utilizzata per aggiungere il contenuto di un valore LOB a un altro valore LOB. |
| ERASE | Procedura utilizzata per eliminare tutto o parte di un valore LOB. |
| TRIM | Procedura utilizzata per eseguire la funzione SQL RTRIM su un valore LOB. |
| COPY | Procedura utilizzata per copiare tutto o parte di un valore LOB da una colonna LOB a un'altra. |

Le procedure e funzioni disponibili all'interno del package DBMS_LOB sono descritte nei paragrafi seguenti nell'ordine in cui sono elencate nella Tabella 27.3. Ulteriori funzioni, specifiche per i tipi di dati BFILE, vengono elencate nella Tabella 27.4.

Tabella 27.4 Procedure e funzioni specifiche per BFILE presenti nel package DBMS_LOB. (*continua*)

| PROCEDURA O FUNZIONE | DESCRIZIONE |
|----------------------|---|
| FILEOPEN | Procedura utilizzata per aprire file per la lettura. |
| FILECLOSE | Procedura utilizzata per chiudere specifici file. Per ogni chiamata FILEOPEN deve esistere una corrispondente chiamata FILECLOSE. |
| FILECLOSEALL | Procedura utilizzata per chiudere tutti i file aperti. |
| FILEEXISTS | Funzione utilizzata per determinare se esiste il file esterno a cui fa riferimento un valore di locatore BFILE. |

Tabella 27.4 Procedure e funzioni specifiche per BFILE presenti nel package DBMS_LOB.

| PROCEDURA O FUNZIONE | DESCRIZIONE |
|----------------------|--|
| FILEGETNAME | Procedura utilizzata per ottenere il nome del file esterno a cui fa riferimento un valore di locatore BFILE. |
| FILEISOPEN | Funzione utilizzata per determinare se il file esterno è aperto. |

Quando si utilizzano BFILE, il massimo numero di file che si possono aprire contemporaneamente è limitato dall'impostazione del parametro SESSION_MAX_OPEN_FILES nel file init.ora del database. L'impostazione predefinita del numero massimo di file BFILE aperti contemporaneamente è pari a 10.

Negli esempi riportati di seguito, la colonna CLOB Proposta_Testo viene utilizzata per visualizzare l'effetto delle procedure e delle funzioni.

READ La procedura READ legge una parte di un valore LOB. Nell'esempio relativo alla tabella PROPOSTA, il testo Proposta_Testo per la prima riga è quello riportato nell'esempio seguente.

```
select Proposta_Testo
  from PROPOSTA
 where Proposta_ID = 1;
```

PROPOSTA_Testo

Questo è il testo di una proposta di azzeramento del campo di PHILLIPS

Questa descrizione è piuttosto breve e avrebbe potuto essere memorizzata in una colonna di tipo VARCHAR2. Poiché tale 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, potendo espandersi sino a 4 GB. Per questi esempi vengono utilizzate brevi stringhe dimostrative. Le stesse funzioni e operazioni possono ugualmente essere eseguite su stringhe CLOB di lunghezza maggiore.

La procedura READ prevede quattro parametri che devono essere specificati nell'ordine riportato di seguito.

1. Locatore LOB.
2. Numero di byte o caratteri da leggere.
3. Offset (punto di partenza della lettura) dall'inizio del valore LOB.
4. Output per i dati 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, che richiede come dato in input il locatore LOB, viene eseguita all'interno di blocchi PL/SQL. Sarà necessario selezionare il valore del locatore LOB, utilizzarlo come dato in ingresso per la procedura READ e visualizzare il testo letto per mezzo del package DBMS_OUTPUT.

Nota sul package DBMS_OUTPUT Come si è ricordato nel Capitolo 24, il package DBMS_OUTPUT può essere utilizzato per visualizzare i valori di variabili all'interno di un blocco PL/SQL. La procedura PUT_LINE all'interno di DBMS_OUTPUT visualizza il risultato specificato su una riga. Prima di utilizzare il package DBMS_OUTPUT è necessario eseguire il comando descritto di seguito:

```
set serveroutput on
```

Nel paragrafo successivo è descritta la creazione di un blocco PL/SQL in grado di leggere dati da un LOB.

Esempio di utilizzo 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 deve essere selezionato dalla tabella che contiene il LOB. Poiché il valore del locator deve essere fornito come dato in input alla procedura READ, è necessario utilizzare variabili PL/SQL per contenere il valore del locator. La procedura READ, a sua volta, inserisce il proprio risultato in una variabile PL/SQL. Il package DBMS_OUTPUT consente di visualizzare il valore in uscita. La struttura del blocco PL/SQL per questo esempio è riportata di seguito;

```
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 var_quanti;
    set value for var_offset;
    select locator value into var_locatore from table;
    DBMS_LOB.READ(var_locatore, var_quanti, var_offset, var_output);
    DBMS_OUTPUT.PUT_LINE('Output:' || var_output);
end;
/
```

NOTA Per ulteriori informazioni sui blocchi PL/SQL e i loro componenti si rimanda al Capitolo 22.

Per la tabella PROPOSTA, la selezione dei primi dieci caratteri della colonna Proposta_Testo fornisce il risultato riportato di seguito.

```
declare
    var_locatore CLOB;
    var_quanti    INTEGER;
    var_offset    INTEGER;
    var_output    VARCHAR2(10);
begin
    var_quanti := 10;
    var_offset := 1;
    select Proposta_Testo into var_locatore
        from PROPOSTA
       where Proposta_ID = 1;
```

```

DBMS_LOB.READ(var_locatore, var_quanti, var_offset, var_output);
DBMS_OUTPUT.PUT_LINE('Inizio del testo della proposta: ' || var_output);
end;
/

```

Eseguendo il precedente blocco PL/SQL si ottiene il risultato seguente:

```
Inizio del testo della proposta: Questo è i
```

```
PL/SQL procedure successfully completed.
```

Il risultato in output visualizza i primi dieci caratteri del valore Proposta_Testo, 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_locatore    CLOB;
    var_quanti      INTEGER;
    var_offset       INTEGER;
    var_output       VARCHAR2(10);

```

Quindi, alle variabili vengono assegnati i relativi valori.

```

begin
    var_quanti := 10;
    var_offset := 1;

```

Il locatore è stato selezionato dalla tabella e memorizzato in una variabile di nome *var_locatore*:

```

select Proposta_Testo into var_locatore
  from PROPOSTA
 where Proposta_ID = 1;

```

In seguito viene richiamata la procedura READ, utilizzando i valori sino a questo momento assegnati.

```
DBMS_LOB.READ(var_locatore, var_quanti, var_offset, var_output);
```

Come risultato dell'esecuzione della procedura READ, la variabile var_output viene riempita con i dati letti. La procedura PUT_LINE visualizza tali dati.

```
DBMS_OUTPUT.PUT_LINE('Inizio del testo della proposta: ' || var_output);
```

Per la restante parte del presente capitolo viene utilizzato questo stesso formato per i blocchi PL/SQL.

Per selezionare una sezione differente dell'oggetto CLOB Proposta_Testo, si modifichino le variabili di quantità e di offset. Se si modifica il numero di caratteri letti, sarà necessario modificare anche le dimensioni della variabile in uscita.

Nell'esempio seguente vengono letti 12 caratteri da Proposta_Testo, a partire dal decimo.

```

declare
    var_locatore    CLOB;
    var_quanti      INTEGER;

```

```

var_offset      INTEGER;
var_output      VARCHAR2(12);
begin
  var_quanti := 12;
  var_offset := 10;
  select Proposta_Testo into var_locatore
    from PROPOSTA
   where Proposta_ID = 1;
  DBMS_LOB.READ(var_locatore, var_quanti, var_offset, var_output);
  DBMS_OUTPUT.PUT_LINE('Parte del testo della proposta: '|| var_output);
end;
/

```

Nell'esempio seguente è riportato il risultato del precedente blocco PL/SQL.

```
Parte del testo della proposta: l testo di u
```

```
PL/SQL procedure successfully completed.
```

In output vengono visualizzati 12 caratteri del valore Proposta_Testo, a partire dal decimo compreso.

Se Proposta_Testo fosse stata creata come una colonna VARCHAR2, sarebbe stato possibile selezionare tali dati utilizzando la funzione SUBSTR. La lunghezza massima per la colonna VARCHAR2 sarebbe però stata pari a 4000 caratteri. Inoltre, si noti 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 con le stesse modalità utilizzate per la selezione da oggetti CLOB. Per gli oggetti BLOB è necessario effettuare il declare del buffer di uscita per utilizzare il tipo di dati RAW. Nei blocchi PL/SQL, i tipi di dati RAW e VARCHAR2 (utilizzati per le variabili in output delle letture CLOB e BLOB) hanno una lunghezza massima pari a 32767.

SUBSTR La funzione SUBSTR all'interno del package DBMS_LOB esegue la funzione SQL SUBSTR su un valore LOB. Tale funzione prevede quattro parametri in input, che devono essere specificati nell'ordine riportato di seguito.

1. Locatore LOB.
2. Numero di byte o caratteri da leggere.
3. Offset (punto di partenza della lettura) dall'inizio del valore LOB.

SUBSTR, essendo una funzione, non restituisce direttamente un valore di variabile in output come avviene con READ. Per la procedura READ si dichiara una variabile in output e quindi la si riempie all'interno della chiamata alla procedura stessa. Il blocco PL/SQL utilizzato per la procedura READ è riportato nell'esempio seguente, con le righe relative alla variabile in uscita evidenziate in grassetto.

```

declare
  var_locatore  CLOB;
  var_quanti    INTEGER;
  var_offset     INTEGER;

```

```

var_output      VARCHAR2(12);
begin
    var_quanti := 12;
    var_offset := 10;
    select Proposta_Testo into var_locatore
        from PROPOSTA
        where Proposta_ID = 1;
    DBMS_LOB.READ(var_locatore, var_quanti, var_offset, var_output);
    DBMS_OUTPUT.PUT_LINE('Parte del testo della proposta: ' || var_output);
end;
/

```

Poiché SUBSTR è una funzione, la variabile in output deve essere riempita in modo differente. Il formato è quello riportato di seguito:

```
var_output := DBMS_LOB.SUBSTR(var_locatore, var_quanti, var_offset);
```

Il seguente blocco PL/SQL utilizza la funzione SUBSTR del package DBMS_LOB per selezionare 12 caratteri dalla colonna Proposta_Testo LOB, a partire dal decimo.

```

declare
    var_locatore  CLOB;
    var_quanti    INTEGER;
    var_offset    INTEGER;
    var_output     VARCHAR2(12);
begin
    var_quanti := 12;
    var_offset := 10;
    select Proposta_Testo into var_locatore
        from PROPOSTA
        where Proposta_ID = 1;
    var_output := DBMS_LOB.SUBSTR(var_locatore, var_quanti, var_offset);
    DBMS_OUTPUT.PUT_LINE('Sottostringa del testo della proposta: ' || var_output);
end;
/

```

Di seguito è riportato un esempio dell'output prodotto:

```
Sottostringa del testo della proposta: l testo di u
```

```
PL/SQL procedure successfully completed.
```

Il formato dell'esempio di SUBSTR relativo al precedente blocco PL/SQL è praticamente identico al formato dell'esempio relativo alla procedura READ (sono inoltre stati selezionati gli stessi dati). Le sole differenze sono rappresentate dal nome della chiamata alla funzione e dalla modalità di assegnamento di un valore alla variabile in output. Come nell'esempio di 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. READ può essere utilizzata per ricavare sottostringhe (come negli esempi precedenti), ma è più spesso utilizzata al-

l'interno di un ciclo. Ad esempio, se il valore LOB fosse stato più ampio della più grande variabile RAW o VARCHAR2 consentita in un blocco PL/SQL (32767 caratteri), sarebbe stato possibile utilizzare la procedura READ all'interno di un ciclo per leggere tutti i dati del valore LOB. Per ulteriori informazioni sui differenti tipi di ciclo utilizzabili in blocchi PL/SQL si rimanda al Capitolo 22.

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 che devono essere specificati nell'ordine riportato di seguito.

1. Locatore LOB.
2. Sequenza da verificare (byte RAW per i BLOB, stringhe di caratteri per i CLOB).
3. Offset (punto di partenza della lettura) dall'inizio del valore LOB.
4. Ricorrenza della sequenza all'interno del valore LOB.

La funzione INSTR all'interno di DBMS_LOB ricerca nel LOB una sequenza specifica di byte o caratteri. La variabile “ricorrenza” consente di specificare quale occorrenza della sequenza all'interno del valore cercato debba essere restituita. Il risultato della funzione INSTR è la posizione di inizio della sequenza all'interno della stringa esaminata. Si consideri l'esempio SQL seguente.

```
INSTR('ABCABC', 'A', 1, 1) = 1
```

All'interno della stringa 'ABCABC', viene cercato il carattere 'A', partendo dalla prima posizione della stringa e cercando la prima ricorrenza. 'A' viene trovata nella prima posizione della stringa esaminata. Iniziando la ricerca dalla seconda posizione, il risultato sarebbe stato il seguente:

```
INSTR('ABCABC', 'A', 2, 1) = 4
```

Dato che questa ricerca INSTR inizia dalla seconda posizione, la prima 'A' non fa parte del valore analizzato. Per questo motivo la prima 'A' trovata sarà quella in quarta posizione.

Se si modifica l'ultimo parametro nella funzione SQL INSTR in modo da cercare la seconda ricorrenza della stringa 'A', partendo dalla seconda posizione, non viene trovato nulla:

```
INSTR('ABCABC', 'A', 2, 2) = 0
```

Se non riesce a trovare la corretta ricorrenza della sequenza cercata, la funzione SQL INSTR restituisce 0. La funzione INSTR del package DBMS_LOB opera nello stesso modo.

Poiché INSTR è una funzione, è necessario assegnare la sua variabile in uscita allo stesso modo utilizzato per assegnare la variabile in output della funzione SUBSTR. Nell'esempio seguente viene analizzato il valore CLOB Proposta_Testo alla ricerca della stringa 'pr'. Per memorizzare il risultato della funzione INSTR viene dichiarata una variabile "var_posizione".

```
declare
```

```

var_locatore    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 Proposta_Testo into var_locatore
    from PROPOSTA
   where Proposta_ID = 1;
  var_posizione := DBMS_LOB.INSTR(var_locatore, var_sequenza, var_offset,
var_occor);
  DBMS_OUTPUT.PUT_LINE('Trovata stringa alla posizione: ' ||
var_posizione);
end;
/

```

Il risultato è il seguente.

```
Trovata stringa alla posizione: 26
```

```
PL/SQL procedure successfully completed.
```

Il risultato mostra che la stringa cercata ‘pr’ viene trovata all’interno del testo Proposta_Testo, a partire dal ventitreesimo carattere del valore LOB.

GETLENGTH La funzione GETLENGTH del package DBMS_LOB restituisce la lunghezza del valore LOB. Questa funzione è simile alla funzione SQL LENGTH, ma viene utilizzata solo per i valori LOB. La funzione SQL LENGTH non può essere utilizzata per i valori LOB.

La funzione GETLENGTH del package DBMS_LOB richiede solo un parametro in input, il valore del locatore per il LOB. Nell’esempio seguente vengono dichiarate variabili per contenere il valore del locatore e il valore in output. La funzione GETLENGTH viene quindi eseguita e il risultato viene visualizzato per mezzo della procedura PUT_LINE.

```

declare
  var_locatore  CLOB;
  var_lunghezza INTEGER;
begin
  select Proposta_Testo into var_locatore
    from PROPOSTA
   where Proposta_ID = 1;
  var_lunghezza := DBMS_LOB.GETLENGTH(var_locatore);
  DBMS_OUTPUT.PUT_LINE('Lunghezza di LOB: ' || var_lunghezza);
end;
/

```

Di seguito è riportato il risultato di GETLENGTH:

Lunghezza di LOB: 70

PL/SQL procedure successfully completed.

Se il valore LOB è NULL, la funzione GETLENGTH restituisce un valore NULL.

COMPARE La funzione COMPARE del package DBMS_LOB confronta due valori LOB. Se i due valori LOB sono identici, la funzione COMPARE restituisce zero, altrimenti restituisce un valore intero differente da zero (normalmente 1 o -1). Per eseguire la funzione COMPARE, il package DBMS_LOB esegue due procedure READ e confronta i risultati. Pertanto, per ciascun valore LOB da confrontare dovrà essere fornito un locatore e un valore di offset. Il numero di caratteri o byte da confrontare è lo stesso per ambedue i valori LOB. Il confronto è possibile solo tra valori LOB dello stesso tipo.

La funzione COMPARE prevede cinque parametri in input, che devono essere specificati nell'ordine riportato di seguito.

1. Locatore LOB per il primo LOB.
2. Locatore LOB per il secondo LOB.
3. Variabile che specifica il numero di byte o caratteri da confrontare.
4. Offset (punto di partenza della lettura) dall'inizio del primo valore LOB.
5. Offset (punto di partenza della lettura) dall'inizio del secondo valore LOB.

Poiché i due LOB da confrontare 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. Nell'esempio seguente, vengono confrontati i primi 25 caratteri dei valori Proposta_Testo di due voci: il record Proposta_ID 1 e il record Proposta_ID 3.

```

declare
    prima_var_locatore    CLOB;
    seconda_var_locatore  CLOB;
    var_quanti            INTEGER;
    prima_var_offset       INTEGER;
    seconda_var_offset     INTEGER;
    var_output             INTEGER;
begin
    var_quanti      := 25;
    prima_var_offset := 1;
    seconda_var_offset := 1;
    select Proposta_Testo into prima_var_locatore
        from PROPOSTA
       where Proposta_ID = 1;
    select Proposta_Testo into seconda_var_locatore
        from PROPOSTA
       where Proposta_ID = 3;
    var_output := DBMS_LOB.COMPARE(prima_var_locatore, seconda_var_locatore,
                                    var_quanti, prima_var_offset, seconda_var_offset);
    DBMS_OUTPUT.PUT_LINE('Valore del confronto (0 se uguali): ' ||
```

```
var_output);
end;
/
```

Di seguito è riportato il risultato:

```
Valore del confronto (0 se uguali): 1
```

```
PL/SQL procedure successfully completed.
```

Se le due stringhe sono identiche, la funzione COMPARE restituisce zero. Nell'esempio successivo vengono confrontati gli stessi due LOB, ma questa volta per il confronto vengono utilizzati solo i primi cinque caratteri. Poiché ambedue i LOB iniziano con i caratteri 'Ques', la funzione COMPARE restituisce 0.

```
declare
    prima_var_locatore    CLOB;
    seconda_var_locatore  CLOB;
    var_quanti            INTEGER;
    prima_var_offset       INTEGER;
    seconda_var_offset     INTEGER;
    var_output             INTEGER;
begin
    var_quanti      := 5;
    prima_var_offset := 1;
    seconda_var_offset := 1;
    select Proposta_Testo into prima_var_locatore
        from PROPOSTA
       where Proposta_ID = 1;
    select Proposta_Testo into seconda_var_locatore
        from PROPOSTA
       where Proposta_ID = 3;
    var_output := DBMS_LOB.COMPARE(prima_var_locatore, seconda_var_locatore,
                                    var_quanti, prima_var_offset, seconda_var_offset);
    DBMS_OUTPUT.PUT_LINE('Valore del confronto(0 se uguali): ' || 
var_output);
end;
/
```

Di seguito è riportato il risultato della funzione COMPARE modificata:

```
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 utilizza la funzione COMPARE su colonne BLOB, è necessario definire le variabili dei locatori come tipi di dati RAW.

WRITE La procedura WRITE del package DBMS_LOB consente la scrittura di dati in locazioni specifiche del LOB. Ad esempio, è possibile scrivere dati binari in una determinata sezione di un BLOB, riscrivendo i dati preesistenti in tale posizio-

ne. Utilizzando la procedura WRITE è possibile scrivere dati alfanumerici nei campi CLOB. Tale procedura prevede quattro parametri in input, che devono essere specificati nell'ordine riportato di seguito.

1. Locatore LOB per il LOB.
2. Variabile che specifica 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 inserire.

Poiché la procedura WRITE aggiorna il valore LOB, è necessario bloccare la riga per mezzo di un comando select for update, come nell'esempio seguente. In questo esempio viene selezionato e bloccato un record. Di seguito viene scritto il testo 'AGGIUNGI TESTO' all'interno del valore LOB, sostituendo i dati a partire dalla posizione 10 (come definito dalla variabile di offset).

```
declare
    var_locatore    CLOB;
    var_quanti      INTEGER;
    var_offset       INTEGER;
    var_buffer       VARCHAR2(12);
begin
    var_quanti := 12;
    var_offset := 10;
    var_buffer := 'AGGIUNGI TESTO';
    select Proposta_Testo into var_locatore
        from PROPOSTA
        where Proposta_ID = 3
    for update;
    DBMS_LOB.WRITE(var_locatore, var_quanti, var_offset, var_buffer);
commit;
end;
/
```

Dato che la procedura WRITE modifica i dati, si aggiunge un comando commit prima della clausola end del blocco PL/SQL.

La procedura WRITE non restituisce risultati in output. Per visualizzare i dati modificati è necessario selezionare nuovamente il valore LOB dalla tabella.

```
select Proposta_Testo
    from PROPOSTA
    where Proposta_ID = 3;
```

```
PROPOSTA_Testo
-----
Questo è AGGIUNGI TEST0a proposta.
```

La procedura WRITE ha riscritto 14 caratteri, a partire dal decimo carattere del testo Proposta_Testo.

APPEND La procedura APPEND del package DBMS_LOB aggiunge i dati di un LOB a un secondo LOB. Poiché ciò corrisponde all'aggiornamento di un valore LOB, il record da aggiornare deve essere bloccato prima di eseguire la procedura APPEND.

La procedura APPEND richiede solo due parametri: il valore del locatore per il LOB di destinazione e il locatore per il LOB sorgente. Se ambedue i parametri sono NULL, la procedura APPEND restituisce un errore.

Nell'esempio seguente vengono definiti due locatori. Il primo, "var_locatore_dest", rappresenta il locatore del LOB a cui aggiungere i dati. Il secondo, "var_locatore_sorg", rappresenta il locatore dei dati sorgente da aggiungere al LOB di destinazione. Quest'ultimo, dovendo essere aggiornato, viene bloccato per mezzo di un comando select for update. Nell'esempio seguente, viene aggiunto il valore Proposta_Testo per il record Proposta_ID 1 al valore Proposta_Testo per il record Proposta_ID 3.

```
declare
    var_locatore_dest    CLOB;
    var_locatore_sorg    CLOB;
begin
    select Proposta_Testo into var_locatore_dest
        from PROPOSTA
       where Proposta_ID = 3
         for update;
    select Proposta_Testo into var_locatore_sorg
        from PROPOSTA
       where Proposta_ID = 1;
    DBMS_LOB.APPEND(var_locatore_dest, var_locatore_sorg);
commit;
end;
/
```

Per visualizzare i dati modificati, si selezioni il record dalla tabella PROPOSTA.

```
select Proposta_Testo
      from PROPOSTA
     where Proposta_ID = 3;
```

PROPOSTA_Testo

Questo è AGGIUNGI TESTOa proposta.Questo è il testo di una proposta di azzeramen

Per default, solo 80 caratteri dei valori LONG e LOB vengono visualizzati durante le selezioni. Per visualizzare il testo restante, occorre utilizzare il comando set long.

```
set long 1000

select Proposta_Testo
      from PROPOSTA
     where Proposta_ID = 3;
```

PROPOSTA_Testo

Questo è AGGIUNGI TEST0a proposta.Questo è il testo di una proposta di azzeramento del campo di PHILLIPS

In questo esempio vengono visualizzati i primi 1000 caratteri del valore LOB. Per sapere quanto è lungo il LOB, occorre utilizzare la funzione GETLENGTH descritta in precedenza nel presente capitolo.

Oltre a consentire di aggiungere valori CLOB, la procedura APPEND consente 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 alla 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 dati RAW (come quelli utilizzati da applicazioni di trasmissione digitale dei dati) al posto di dati binari formattati (come file di fogli elettronici formattati da un programma).

ERASE La procedura ERASE del package DBMS_LOB consente di eliminare caratteri o byte in qualsiasi punto di un LOB; può essere utilizzata per eliminare l'intero valore LOB o una determinata sezione di questo. La procedura ERASE è simile per funzionamento alla procedura WRITE. Se si eliminano dei dati in un valore BLOB, tali dati 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 eliminata una parte del testo Proposta_Testo per il record con Proposta_ID 3. L'eliminazione dei caratteri inizia con offset pari a 10 e prosegue per 25 caratteri. La procedura ERASE prevede tre parametri in input che devono essere specificati nell'ordine riportato di seguito.

1. Locatore LOB per il LOB.
2. Variabile che specifica il numero di byte o caratteri da eliminare.
3. Offset (punto iniziale di eliminazione) dall'inizio del valore LOB.

```

declare
    var_locatore  CLOB;
    var_quanti    INTEGER;
    var_offset     INTEGER;
begin
    var_quanti := 25;
    var_offset := 10;
    select Proposta_Testo into var_locatore
        from PROPOSTA
       where Proposta_ID = 3
         for update;
    DBMS_LOB.ERASE(var_locatore, var_quanti, var_offset);
commit;
end;
/

```

Per visualizzare la modifica al record è sufficiente eseguire una query del valore Proposta_Testo.

```
select Proposta_Testo
  from PROPOSTA
 where Proposta_ID = 3;
```

PROPOSTA_Testo

Questo è i roposta.Questo è il testo di una proposta di azzeramento del campo di PHILLIPS

NOTA *Lo spazio occupato in precedenza dai dati eliminati viene riempito con spazi vuoti. La posizione dei restanti dati del LOB non cambia.*

TRIM La procedura TRIM del package DBMS_LOB consente di ridurre le dimensioni di un valore LOB eliminando caratteri o byte alle estremità del valore LOB stesso (come la funzione SQL RTRIM). Per eseguire la procedura TRIM è necessario specificare il locatore per il LOB e la nuova lunghezza del LOB stesso.

Nell'esempio seguente, vengono rimossi i primi dieci caratteri del testo Proposta_Testo per il record Proposta_ID 3.

Dato che la procedura TRIM modifica i dati, viene aggiunto un comando commit prima della clausola end del blocco PL/SQL.

```
declare
  var_locatore      CLOB;
  nuova_var_lung    INTEGER;
begin
  nuova_var_lung   := 10;
  select Proposta_Testo into var_locatore
    from PROPOSTA
   where Proposta_ID = 3
     for update;
  DBMS_LOB.TRIM(var_locatore, nuova_var_lung);
commit;
end;
/
```

La procedura TRIM non restituisce dati in output. Per visualizzare i dati modificati occorre selezionare nuovamente il valore LOB dalla tabella.

```
select Proposta_Testo
  from PROPOSTA
 where Proposta_ID = 3;
```

PROPOSTA_Testo

Questo è i

Nell'esempio viene visualizzato il risultato della procedura TRIM successiva alla procedura ERASE eseguita nel paragrafo precedente.

COPY La procedura COPY del package DBMS_LOB consente la copia di 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 copia è possibile specificare l'offset di lettura e l'offset di scrittura. La procedura COPY prevede cinque parametri che devono essere specificati nell'ordine riportato di seguito.

1. Locatore del LOB di destinazione.
2. Locatore del LOB sorgente.
3. Variabile che specifica 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 sorgente.

La procedura COPY combina le capacità delle procedure READ e WRITE. Come avviene per la procedura WRITE, la procedura COPY modifica il valore LOB. Per questo motivo è necessario prima bloccare il record. Inoltre il blocco PL/SQL deve contenere un comando commit. Nell'esempio seguente, i primi 55 caratteri del record Proposta_ID 1 vengono copiati nel record Proposta_ID. Dato che il LOB di destinazione utilizza un offset pari a 1, i dati copiati riscrivono i dati preesistenti nel LOB di destinazione.

```
declare
    var_locatore_dest    CLOB;
    var_locatore_sorg     CLOB;
    var_quanti            INTEGER;
    var_offset_dest        INTEGER;
    var_offset_sorg        INTEGER;
begin
    var_quanti      := 55;
    var_offset_dest  := 1;
    var_offset_sorg  := 1;
    select Proposta_Testo into var_locatore_dest
        from PROPOSTA
       where Proposta_ID = 3
         for update;
    select Proposta_Testo into var_locatore_sorg
        from PROPOSTA
       where Proposta_ID = 1;
    DBMS_LOB.COPY(var_locatore_dest, var_locatore_sorg,
                  var_quanti, var_offset_dest, var_offset_sorg);
commit;
end;
/
```

La procedura COPY non restituisce risultati in output. Per visualizzare i dati modificati occorre selezionare nuovamente il valore LOB dalla tabella.

```
select Proposta_Testo
  from PROPOSTA
 where Proposta_ID = 3;
```

PROPOSTA_Testo

Questo è il testo di una proposta di azzeramento del campo di PHILLIPS

Utilizzo delle funzioni e procedure BFILE Dato che i valori BFILE vengono 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, riportate in precedenza nella Tabella 27.4, 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 di controllo dell'esistenza è necessaria in quanto ORACLE non conserva i dati memorizzati nei file esterni, ma solo i puntatori creati mediante la procedura BFILENAME al momento dell'inserimento o dell'aggiornamento dei record. Nella Tabella 27.5 sono riportate le procedure e le funzioni relative ai valori BFILE, insieme ai rispettivi parametri di input e output.

Tabella 27.5 Parametri di input e output per procedure e funzioni BFILE.

| PROCEDURA O FUNZIONE | PARAMETRI IN INPUT | PARAMETRI IN OUTPUT |
|----------------------|----------------------------|----------------------------|
| FILECLOSE | Locatore file | Nessuno |
| FILECLOSEALL | Nessuno | Nessuno |
| FILEEXISTS | Locatore file | Valore intero |
| FILEGETNAME | Locatore file | Alias directory, nome file |
| FILEISOPEN | Locatore file | Valore intero |
| FILEOPEN | Locatore file, modo aperto | Nessuno |

Le procedure e funzioni elencate nella Tabella 27.5 consentono il controllo dei file utilizzati dai blocchi PL/SQL. Ad esempio, per leggere i primi dieci byte dal BFILE Lettera_Present per il record Proposta_ID 2 è necessario aprire il file ed eseguire la procedura READ. I dati letti potranno quindi essere visualizzati mediante la procedura PUT_LINE del package DBMS_OUTPUT.

Se nel blocco PL/SQL si utilizza una sezione di 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 22.

Eliminazione dei LOB

Quando si eliminano valori LOB, viene eliminato il valore del locatore. Se il valore eliminato è un LOB interno (BLOB o CLOB o NCLOB), vengono cancellati sia il locatore, che il valore LOB. Se il valore eliminato è un LOB esterno (BFILE), viene eliminato solo il valore del locatore. Il file a cui punta il locatore deve, se necessario, essere eliminato manualmente.

• Capitolo 28

• **Gli snapshot**

- 28.1 **Che cosa sono gli snapshot?**
- 28.2 **Privilegi di sistema richiesti**
- 28.3 **Privilegi richiesti per la tabella**
- 28.4 **Snapshot semplici e snapshot complessi**
- 28.5 **Snapshot di sola lettura e snapshot aggiornabili**
- 28.6 **Sintassi del comando create snapshot**
- 28.7 **Aggiornamento degli snapshot**
- 28.8 **Snapshot e trigger**
- 28.9 **Sintassi per il comando create snapshot log**
- 28.10 **Visualizzazione di informazioni su snapshot esistenti**
- 28.11 **Modifica di snapshot e log di snapshot**
- 28.12 **Scaricamento di snapshot e log di snapshot**

Per migliorare le prestazioni di un'applicazione che utilizza dati distribuiti, è possibile creare copie locali di tabelle remote. ORACLE mette a disposizione gli *snapshot* come mezzo per gestire le copie locali di tabelle remote.

Gli snapshot possono essere utilizzati per replicare un'intera tabella o parte di essa, oppure per replicare il risultato di una query eseguita su più tabelle. L'aggiornamento dei dati replicati può essere effettuato automaticamente dal database a intervalli di tempo definibili.

Nel presente capitolo sono descritti i comandi e i privilegi necessari per creare e gestire gli snapshot, insieme ad alcuni esempi pratici.

Per utilizzare gli snapshot è necessario che nel database di ORACLE sia installata l'opzione distribuita. Tale opzione può essere installata mediante il programma d'installazione di ORACLE, oppure dopo che il database è stato creato.

28.1 Che cosa sono gli snapshot?

Gli snapshot sono copie (dette anche *repliche*) di dati remoti, basate su query. Nella loro forma più semplice, gli snapshot possono essere pensati come tabelle create da un comando come quello riportato nel seguente esempio.

```
create table REGISTRO_LOCALE  
as  
select * from REGISTRO@REMOTE_CONNECT;
```

In questo esempio, nel database locale viene creata una tabella REGISTRO_LOCALE, che viene riempita con dati provenienti da un database remoto (definito dal link di database REMOTE_CONNECT). Una volta creata la tabella REGISTRO_LOCALE, i suoi dati possono però risultare immediatamente fuori sincronia rispetto alla tabella master (REGISTRO@REMOTE_CONNECT). Inoltre, essendo una tabella locale, REGISTRO_LOCALE potrebbe essere aggiornata da utenti locali e in tal modo risulterebbe ulteriormente compromessa la sua sincronizzazione con la tabella master.

Nonostante questi problemi di sincronia, la replicazione dei dati eseguita in questo modo porta notevoli vantaggi. La creazione di copie locali di dati remoti può essere utile per migliorare le prestazioni di query distribuite, in particolare se i dati delle tabelle non cambiano molto spesso. Gli snapshot sono inoltre utili per ambienti di supporto decisionale, nei quali complesse query vengono utilizzate per effettuare il “riepilogo” periodico dei dati in tabelle riassuntive da utilizzare durante le analisi.

ORACLE consente di gestire la sincronizzazione della tabella master per mezzo degli snapshot. Utilizzando gli snapshot viene fissato un *intervallo di aggiornamento* per la programmazione dell'aggiornamento dei dati replicati. Gli aggiornamenti locali possono essere evitati ed è possibile utilizzare aggiornamenti basati sulle transazioni. Gli aggiornamenti basati sulle transazioni, disponibili per alcuni tipi di snapshot, inviano dal database master solo le righe che sono cambiate per lo snapshot. Questa capacità, descritta nei paragrafi “Snapshot semplici e complessi” e “Sintassi per il comando create snapshot log” più avanti nel presente capitolo, può migliorare significativamente le prestazioni degli aggiornamenti agli snapshot.

28.2 Privilegi di sistema richiesti

Per creare uno snapshot è necessario disporre dei privilegi richiesti per la creazione di oggetti sottostanti, che verranno utilizzati dallo snapshot stesso. Servono i privilegi di sistema CREATE SNAPSHOT, CREATE TABLE, CREATE VIEW e CREATE INDEX (solo per gli snapshot semplici). Inoltre è necessario disporre del privilegio di sistema UNLIMITED TABLESPACE o di una parte di spazio identificata e sufficiente in un tablespace locale.

Gli oggetti sottostanti utilizzati dallo snapshot sono memorizzati nell'account dell'utente (detto anche *schema*). Se lo snapshot deve essere creato in uno schema differente dal proprio, è necessario disporre dei privilegi di sistema CREATE ANY

SNAPSHOT, CREATE ANY TABLE, CREATE ANY VIEW e CREATE ANY INDEX (solo per gli snapshot semplici). Inoltre è necessario disporre del privilegio di sistema UNLIMITED TABLESPACE o di una parte di spazio identificata e sufficiente in un tablespace locale.

Gli snapshot di tabelle remote richiedono query su tabelle remote. Per questo motivo è necessario disporre di privilegi che consentano l'utilizzo di un link di database che acceda al database remoto. Il link utilizzato può essere pubblico o privato. Se è privato, è necessario disporre del privilegio di sistema CREATE DATABASE LINK. Per ulteriori informazioni sui link di database si rimanda al Capitolo 21.

28.3 Privilegi richiesti per la tabella

Per la creazione di uno snapshot è necessario fare riferimento a tabelle in un database remoto, per mezzo di un link di database. L'account utilizzato dal link nel database remoto deve avere accesso alle tabelle e alle viste utilizzate dal link stesso. Non è possibile creare uno snapshot basato su oggetti di proprietà dell'utente SYS.

All'interno del database locale è possibile concedere il privilegio SELECT per il proprio snapshot ad altri utenti locali. Poiché molti snapshot sono di sola lettura (sono disponibili anche snapshot aggiornabili) non sono necessari ulteriori privilegi. Se si desidera creare uno snapshot aggiornabile, è necessario garantire ai propri utenti il privilegio UPDATE sia sullo snapshot, sia sulla sottostante tabella locale a cui esso accede. Per ulteriori informazioni sugli oggetti locali creati dagli snapshot si rimanda al paragrafo “Oggetti locali e remoti” più avanti in questo capitolo.

28.4 Snapshot semplici e snapshot complessi

Le query che formano la base degli snapshot sono raggruppate in due categorie: *semplici* e *complesse*. Uno snapshot che utilizza una query semplice (descritta più avanti), è uno snapshot semplice. Uno snapshot che utilizza una query complessa (ovvero una query che non è possibile qualificare come semplice), è uno snapshot complesso. La distinzione tra snapshot semplici e complessi influenza sulle opzioni disponibili durante gli aggiornamenti degli snapshot. Tali opzioni sono descritte nel paragrafo “Aggiornamento degli snapshot”, più avanti in questo capitolo. È però importante tenere presente tali distinzioni nel momento in cui si prende in considerazione per la prima volta l'utilizzo degli snapshot.

Una query semplice possiede le caratteristiche seguenti.

- Seleziona righe da una sola tabella (non da una vista).
- Non effettua operazioni di impostazione, unioni, group by o connect by.

Il risultato di una query semplice contiene record, ciascuno dei quali corrisponde a un singolo record della tabella master. Uno snapshot semplice non deve necessariamente contenere tutti i record della tabella master.

Per limitare le righe restituite è possibile utilizzare una clausola where. Uno snapshot che utilizza una query semplice è uno snapshot semplice.

Uno snapshot complesso è uno snapshot la cui query viola una delle regole specificate per uno snapshot semplice.

Pertanto, qualsiasi snapshot la cui query contenga una sottoquery o una clausola group by è, per definizione, uno snapshot complesso.

Quando si crea uno snapshot non è necessario specificare il tipo (semplice o complesso) di snapshot creato. ORACLE esamina la query dello snapshot e determina il tipo di snapshot e le opzioni di aggiornamento predefinite disponibili per quest'ultimo. Tali opzioni possono essere ridefinite, come descritto nel paragrafo “Sintassi per il comando create snapshot”.

28.5 Snapshot di sola lettura e snapshot aggiornabili

Uno snapshot di sola lettura non è in grado, in sé, di restituire alla tabella master le modifiche ai dati. Uno snapshot aggiornabile è in grado di inviare modifiche alla sua tabella master.

Anche se la distinzione può apparire semplice, le differenze tra questi due tipi di snapshot non sono banali. Uno snapshot di sola lettura viene implementato come un comando create table as select, effettuato tramite il collegamento a un database. Quando si eseguono delle transazioni, queste intervengono solo all'interno della tabella master. Tali transazioni vengono in seguito inviate allo snapshot di sola lettura. In questo modo, il metodo con cui vengono modificate le righe nello snapshot viene mantenuto sotto controllo. Le righe dello snapshot cambiano solo in seguito a una modifica alla tabella master dello snapshot stesso.

In uno snapshot aggiornabile, non vi è controllo sul metodo con cui le righe dello snapshot vengono modificate. Le righe possono essere modificate in seguito a modifiche alla tabella master o direttamente da utenti dello snapshot. Per questo motivo è necessario essere in grado di inviare record dal master allo snapshot e dallo snapshot al master.

Poiché modifiche provengono da più parti, anche i master sono molteplici (a ciò fa riferimento, nella documentazione di ORACLE, il termine *configurazione multi-master*).

Se si utilizzano snapshot aggiornabili, è necessario trattare lo snapshot come un master, completo di tutte le strutture di replicazione sottostanti e di tutte le caratteristiche normalmente presenti nei siti master. È inoltre necessario decidere come i record vengano propagati dallo snapshot verso il master. Durante il trasferimento di record dallo snapshot al master è necessario decidere come risolvere i conflitti. Che cosa succede, ad esempio, se il record con ID=1 viene eliminato dal sito dello snapshot, mentre nel sito master viene creato, in una tabella separata, un record che fa riferimento (mediante una chiave esterna) al record ID=1? Non è possibile eliminare il record ID=1 dal sito master, in quanto tale record ha un record “figlio” che vi fa riferimento. Non è possibile inserire il record figlio nel sito dello snapshot, in quanto il record padre (ID=1) è stato eliminato. Com'è possibile risolvere conflitti di questo genere?

Gli snapshot di sola lettura consentono di evitare la necessità di risolvere conflitti, forzando tutte le transazioni ad avvenire nella tabella master controllata. Ciò può porre dei limiti alla funzionalità, ma è una soluzione adeguata per la maggioranza delle esigenze di replicazione. Se si necessita della replicazione multimaster, è consigliabile consultare la guida *ORACLE Server Replication* per avere alcune linee guida e istruzioni dettagliate per l'implementazione.

28.6 Sintassi del comando **create snapshot**

La sintassi per la creazione di uno snapshot è riportata nel listato seguente. Di seguito alla descrizione del comando sono riportati esempi di utilizzo dello stesso.

```
create snapshot [utente.]snapshot
  { { pctfree intero
      | pctused intero
      | initrans intero
      | maxtrans intero
      | tablespace tablespace
      | storage storage}
    | cluster cluster (colonna [, colonna] ...) } ...
  [using {index [ pctfree intero
                 | pctused intero
                 | initrans intero
                 | maxtrans intero ]
          | [ default] [ master | local] rollback segment
          | segmento rollback] } ]
  [refresh [fast | complete | force]
          [start with data] [next data]
          [with {primary key | rowid}] ]
  [for update] as query
```

Nel comando `create snapshot` esistono quattro sezioni distinte. La prima è il titolo, in cui viene assegnato un nome allo snapshot (la prima riga dell'esempio).

```
create snapshot [utente.]snapshot
```

Lo snapshot viene creato nell'account dell'utente (schema), a meno che non venga specificato un differente nome di utente nel titolo. Nella seconda sezione vengono impostati i parametri di memorizzazione per lo snapshot.

```
{ { pctfree intero
    | pctused intero
    | initrans intero
    | maxtrans intero
    | tablespace tablespace
    | storage storage}
  | cluster cluster (colonna [, colonna] ...) } ...
  [using {index [ pctfree intero
                 | pctused intero
                 | initrans intero]
```

```
| maxtrans intero ]
| [ default] [ master | local] rollback segment
[segmento_rollback] } }
```

I parametri di memorizzazione vengono applicati a una tabella creata nel database locale allo scopo di memorizzare i dati replicati. Per ulteriori informazioni sui parametri di memorizzazione disponibili, si consulti il paragrafo dedicato alla memorizzazione nella Guida alfabetica di riferimento (Capitolo 37).

NOTA È possibile specificare i parametri di memorizzazione che devono essere utilizzati per l'indice che viene creato automaticamente sullo snapshot. È inoltre possibile specificare il segmento di rollback da utilizzare durante la creazione dello snapshot e le operazioni di aggiornamento. La possibilità di specificare un segmento di rollback consente un migliore supporto a transazioni di grandi dimensioni che possono intervenire durante le operazioni sugli snapshot.

Nella terza sezione sono impostate le opzioni di aggiornamento ed è possibile specificare se lo snapshot è basato su ROWID o su chiave primaria.

```
[refresh [fast | complete | force]
[ start with dati] [next dati]
[with {primary key | rowid}] ]
```

L'opzione refresh specifica il meccanismo che ORACLE deve utilizzare per l'aggiornamento dello snapshot. Le tre opzioni disponibili sono: fast, complete e force. Gli aggiornamenti rapidi (fast) sono disponibili solo per gli snapshot semplici e utilizzano tabelle dette *log di snapshot* per inviare specifiche righe dalla tabella master allo snapshot. Gli aggiornamenti completi (complete) ricreano completamente lo snapshot. L'opzione force fa sì che ORACLE utilizzi un aggiornamento rapido se questo è disponibile, un aggiornamento completo altrimenti. Se si è creato uno snapshot semplice, ma si desidera eseguire aggiornamenti completi, è necessario specificare refresh complete nel comando create snapshot. Le opzioni di aggiornamento vengono ulteriormente descritte nel paragrafo “Aggiornamento degli snapshot”, più avanti in questo capitolo. All'interno di questa sezione del comando create snapshot è inoltre possibile specificare il meccanismo utilizzato per correlare i valori nello snapshot ai valori della tabella master, ovvero stabilire se devono essere utilizzati valori RowID o valori della chiave primaria. Per default vengono utilizzate le chiavi principali.

La quarta sezione del comando create snapshot contiene la query utilizzata dallo snapshot:

```
[for update] as query
```

È possibile creare snapshot di tabelle che si trovano all'interno del database locale (ad esempio per automatizzare un programma di aggiornamento), ma gli snapshot sono normalmente creati basandosi su query eseguite su tabelle remote. Se si specifica for update, lo snapshot risulta aggiornabile; in caso contrario è di sola lettura. La maggior parte degli snapshot è una replicazione di sola lettura dei dati master. Se si utilizzano snapshot aggiornabili è necessario essere consapevoli dei problemi come la replicazione a due vie delle modifiche e la risoluzione di conflitti

nelle modifiche ai dati. Gli snapshot aggiornabili sono un esempio di replicazione multimaster. Per ulteriori informazioni sull'implementazione di un ambiente di replicazione multimaster, si rimanda alla guida *ORACLE Server Replication*.

La query che forma la base per lo snapshot non deve utilizzare le pseudocolonne User o SysDate. Nell'esempio seguente viene creato uno snapshot di sola lettura denominato REGISTRO_LOCALE in un database locale, basato su una tabella remota REGISTRO, accessibile tramite il link di database REMOTE_CONNECT.

```
create snapshot REGISTRO_LOCALE
storage (initial 100K next 100K pctincrease 0)
tablespace SNAPS
refresh fast
start with SysDate next SysDate+7
with ROWID
as
select * from REGISTRO@REMOTE_CONNECT;
```

Il comando dell'esempio precedente creerà uno snapshot semplice di sola lettura di nome REGISTRO_LOCALE. La tabella sottostante verrà creata con i parametri di memorizzazione specificati, in un tablespace di nome SNAPS. Visto che i dati nella tabella base locale dello snapshot verranno modificati nel tempo, è normalmente consigliata la memorizzazione degli snapshot in un tablespace a essi dedicato (in questo esempio SNAPS). Poiché l'esempio crea uno snapshot semplice, viene specificata l'opzione di aggiornamento fast. La query dello snapshot specifica che l'intera tabella REGISTRO, senza modifiche, dovrà essere copiata nel database locale. Non appena lo snapshot REGISTRO_LOCALE viene creato, la tabella sottostante viene popolata con i dati di REGISTRO. Da quel momento in poi lo snapshot verrà aggiornato ogni sette giorni. I parametri di memorizzazione che non sono specificati utilizzano i valori predefiniti in relazione al tablespace SNAPS.

Nell'esempio seguente viene creato in un database locale uno snapshot complesso di nome REGISTRO_LOCALE_TOTALI, basato su una tabella remota di nome REGISTRO in un database cui si accede per mezzo del link di database REMOTE_CONNECT. Le principali differenze tra questo snapshot e lo snapshot REGISTRO_LOCALE sono evidenziate in grassetto.

```
create snapshot REGISTRO_LOCALE_TOTALI
storage (initial 50K next 50K pctincrease 0)
tablespace SNAPS
refresh complete
start with SysDate next SysDate+7
as
select Persona, Azione, SUM(Importo) Somma_Importo
  from REGISTRO@REMOTE_CONNECT
  group by Persona, Azione;
```

La query nell'esempio relativo a REGISTRO_LOCALE_TOTALI raggruppa i valori Importo per valori Persona e valori Azione (nella tabella REGISTRO i valori Azione sono 'COMPRATO', 'VENDUTO', 'PAGATO' e 'RICEVUTO').

I due esempi riportati nel presente paragrafo presentano numerosi aspetti importanti.

1. La query group by utilizzata nello snapshot REGISTRO_LOCALE_TOTALI può essere eseguita in SQLPLUS sullo snapshot REGISTRO_LOCALE. Ciò significa che l'operazione group by può essere eseguita all'esterno dello snapshot.
2. REGISTRO_LOCALE_TOTALI, in quanto utilizza una clausola group by, è uno snapshot complesso. Pertanto possono essere utilizzati solamente aggiornamenti completi. REGISTRO_LOCALE, essendo uno snapshot semplice, può utilizzare gli aggiornamenti rapidi.

I due snapshot descritti negli esempi precedenti fanno riferimento alla stessa tabella. Poiché uno dei due è uno snapshot semplice che replica tutte le colonne e le righe della tabella master, il secondo può a prima vista apparire ridondante. In effetti però vi sono casi in cui il secondo snapshot è il più utile dei due.

In primo luogo è necessario ricordare che gli snapshot vengono utilizzati per soddisfare le esigenze di query degli utenti locali. Se tali utenti eseguono sempre operazioni group by nelle proprie query, e le loro colonne di raggruppamento sono fisse, allora per essi può dimostrarsi più utile REGISTRO_LOCALE_TOTALI. In secondo luogo, se il volume di transazioni sulla tabella master REGISTRO è molto alto, o se la tabella master REGISTRO è di dimensioni molto ridotte, non potranno esservi significative differenze nei tempi di aggiornamento degli aggiornamenti rapidi e di quelli completi. Lo snapshot più appropriato è quello che si dimostra più produttivo per gli utenti.

Snapshot basati su ROWID e su valori di chiave primaria

In ORACLE8 è possibile basare i propri snapshot su valori di chiave primaria della tabella master, invece che sui ROWID della stessa. Per decidere quale opzione scegliere, è necessario considerare più fattori.

- *Stabilità del sistema.* Se il sito master non è stabile, può rendersi necessario effettuare prelievi del database che coinvolgano la tabella master. Quando si utilizzano le utilità di esportazione e importazione di ORACLE per effettuare prelievi, è molto probabile che i valori ROWID delle righe vengano modificati. Se il sistema richiede frequenti operazioni di esportazione e importazione, è consigliato l'utilizzo di snapshot basati su valori di chiave primaria.
- *Numero dei dati replicati.* Se normalmente non si desidera replicare le colonne di chiave primaria, è possibile ridurre il numero dei dati replicati, replicando invece i valori ROWID.
- *Numero dei dati inviati durante gli aggiornamenti.* Durante gli aggiornamenti gli snapshot basati sui valori ROWID richiedono normalmente l'invio di una mole di dati minore rispetto agli snapshot basati su valori di chiave primaria (a meno che la chiave primaria sia una colonna molto breve).
- *Dimensioni della tabella log di snapshot.* ORACLE consente la memorizzazione delle modifiche alle tabelle principali in tabelle separate, dette *log di snapshot* (descritte dettagliatamente più oltre nel presente capitolo). Se la chiave primaria consiste di più colonne, il log per uno snapshot basato su valori di chiave primaria po-

trà essere di dimensioni considerevolmente maggiori rispetto al log per un analogo snapshot basato su valori di ROWID.

- *Integrità referenziale.* Per utilizzare snapshot basati 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, è necessario utilizzare snapshot basati su valori ROWID.

Oggetti locali e remoti

Quando si crea uno snapshot, un certo numero di oggetti vengono creati nel database locale e nel database remoto. Gli oggetti di supporto creati all'interno di un database sono gli stessi per snapshot semplici e per snapshot complessi. Con gli snapshot semplici si ha la possibilità di creare oggetti aggiuntivi detti *log di snapshot*, che verranno descritti nel paragrafo “Sintassi per il comando create snapshot log” più oltre nel presente capitolo.

Si consideri lo snapshot semplice descritto nel paragrafo precedente:

```
create snapshot REGISTRO_LOCALE
storage (initial 100K next 100K pctincrease 0)
tablespace SNAPS
refresh fast
start with SysDate next SysDate+7
with ROWID
as
select * from REGISTRO@REMOTE_CONNECT;
```

All'interno del database locale questo comando crea nello schema del proprietario degli snapshot gli oggetti elencati di seguito.

- Una tabella SNAP\$_REGISTRO_LOCALE che è la *tabella base locale* per lo snapshot della tabella remota. Questa tabella contiene i dati replicati, più una colonna supplementare per i valori ROWID master (dato che è stata specificata la clausola *with ROWID*).
- Una vista REGISTRO_LOCALE accessibile agli utenti dello snapshot locale.
- Una vista MVIEW\$_REGISTRO_LOCALE che viene utilizzata dal database durante gli aggiornamenti. Tale vista viene utilizzata solo dal RDBMS, mai dagli utenti.
- Se lo snapshot è semplice, ORACLE creerà un indice sulla tabella base locale dello snapshot (SNAP\$_REGISTRO_LOCALE). Per ulteriori informazioni si rimanda al paragrafo “Indicizzazione delle tabelle di snapshot”.

NOTA Poiché ORACLE utilizza il nome dello snapshot nel nome degli oggetti di supporto, il nome stesso dovrebbe essere mantenuto più breve di 19 caratteri.

Esiste solo una modifica ammessa agli oggetti sottostanti: la tabella SNAP\$_REGISTRO_LOCALE dovrebbe essere indicizzata in modo da rispecchiare i percorsi di query normalmente utilizzati dagli utenti locali.

Se si indicizza la tabella base locale dello snapshot, è necessario tenere conto dei requisiti di memorizzazione degli indici quando si valutano le esigenze di spazio dello snapshot. Per ulteriori informazioni si rimanda al paragrafo “Indicizzazione delle tabelle di snapshot”, più oltre nel presente capitolo.

Quando si effettua una query allo snapshot REGISTRO_LOCALE, in quanto utente del database locale, si sta in effetti rivolgendo una query alla vista REGISTRO_LOCALE della tabella SNAP\$_REGISTRO_LOCALE. Questo livello è stato introdotto per evitare l'accesso diretto alla tabella sottostante. Se si utilizzano snapshot in modalità di sola lettura, ORACLE evita efficacemente l'aggiornamento della tabella base locale dello snapshot.

Dato che REGISTRO_LOCALE è una vista, non è possibile creare un ulteriore snapshot semplice basato su di essa.

Nel database remoto non vengono creati oggetti di supporto, a meno che non vengano utilizzati log di snapshot per registrare le modifiche nella tabella master. I log di snapshot sono descritti nel paragrafo “Sintassi per il comando create snapshot log” del presente capitolo.

Assegnazione dei nomi agli snapshot Poiché ORACLE utilizza il nome dello snapshot come base per i nomi degli oggetti di supporto, il nome stesso non può superare i 19 caratteri di lunghezza, altrimenti i nomi dei relativi oggetti di supporto supererebbero la lunghezza massima di 30 caratteri consentita da ORACLE.

Il nome assegnato a uno snapshot utilizzato da un'applicazione dovrebbe rendere ovvio all'amministratore dell'applicazione il fatto che viene utilizzato uno snapshot. Nell'esempio riportato nel paragrafo precedente, lo snapshot si chiamava REGISTRO_LOCALE, il che implicava l'esistenza in qualche luogo di una copia remota della tabella REGISTRO. Poiché il nome contiene il prefisso generico “LOCAL”, nulla viene implicato che riguardi il contenuto. Se lo snapshot applica alcune restrizioni alle righe o alle colonne restituite, tali azioni dovrebbero essere rispecchiate dal nome dello snapshot stesso. Ad esempio, uno snapshot semplice della tabella REGISTRO che duplichi solo i dati dell'anno in corso potrà essere chiamato REGISTRO_LOCALE_AN_CORR.

Per utilizzare la stessa applicazione su più database, i nomi degli oggetti database dovrebbero essere gli stessi. Per questo motivo è possibile creare nel proprio database locale un sinonimo che punti allo snapshot. Nell'esempio seguente viene creato un sinonimo privato per lo snapshot REGISTRO_LOCALE_AN_CORR di proprietà dell'utente Talbot.

```
create synonym REGISTRO for Talbot.REGISTRO_LOCALE_AN_CORR;
```

Un utente locale di tale sinonimo farà riferimento allo snapshot REGISTRO_LOCALE_AN_CORR ogniqualvolta verrà fatto riferimento a REGISTRO.

Indicizzazione delle tabelle di snapshot Come riportato in precedenza, la tabella SNAP\$_nomesnapshot, detta *tabella base locale*, contiene i dati replicati da un sito remoto (per lo snapshot REGISTRO_LOCALE, la tabella base locale è SNAP\$_REGISTRO_LOCALE). Poiché tali dati sono stati replicati con uno scopo preciso (normalmente il miglioramento delle prestazioni del database o della rete), è importante proseguire in direzione di tale scopo una volta che lo snapshot è stato

creato. I miglioramenti nelle prestazioni vengono normalmente ottenuti per mezzo degli indici. Le colonne che vengono utilizzate di frequente nelle clausole where delle query dovrebbero esser indicizzate. Se nelle query si accede frequentemente a una determinata serie di colonne, può essere creato un indice concatenato su di esse. Per ulteriori informazioni sull'indicizzazione si rimanda al Capitolo 19.

Quando viene creato uno snapshot semplice, ORACLE replica i dati della tabella sorgente. Se lo snapshot viene creato con la clausola with ROWID, i valori ROWID verranno memorizzati nella tabella base locale in una colonna di nome M_ROW\$\$ e su tale colonna verrà posto un vincolo di unicità. L'indice per tale vincolo avrà lo stesso nome del vincolo stesso. Se lo snapshot viene creato con la clausola with primary key, e tutte le colonne di chiave primaria della tabella vengono replicate, i valori di chiave primaria vengono indicizzati nella tabella base locale.

ORACLE non crea automaticamente indici per gli snapshot complessi. Tali indici devono essere creati manualmente. Per creare indici sulla tabella base locale, occorre utilizzare il comando `create index` (si rimanda alla Guida alfabetica di riferimento del Capitolo 37). Non si deve creare alcun vincolo sulla tabella base locale dello snapshot.

Poiché non vengono creati indici sulle colonne a cui gli utenti probabilmente rivolgeranno query dallo snapshot, è necessario creare indici sulla tabella base locale dello snapshot.

28.7 Aggiornamento degli snapshot

I dati possono essere replicati negli snapshot sia una volta sola (alla creazione), sia a intervalli regolari. Il comando `create snapshot` consente di impostare l'intervalle di aggiornamento, delegando al database la responsabilità di tali aggiornamenti. Nei paragrafi successivi viene descritta l'esecuzione di aggiornamenti manuali e automatici.

Aggiornamenti automatici

All'interno del comando `create snapshot` è possibile specificare la pianificazione degli aggiornamenti dello snapshot. Si prenda in considerazione lo snapshot REGISTRO_LOCALE descritto in precedenza. Le impostazioni relative agli intervalli di aggiornamento, definite per mezzo del comando `create snapshot`, sono riportate in grassetto:

```
create snapshot REGISTRO_LOCALE
storage (initial 100K next 100K pctincrease 0)
tablespace SNAPS
refresh fast
start with SysDate next SysDate+7
with ROWID
as
select * from REGISTRO@REMOTE_CONNECT;
```

Nella pianificazione degli aggiornamenti sono presenti tre componenti. In primo luogo viene specificato il tipo di aggiornamento (fast, complete o force). Gli aggiornamenti rapidi utilizzano tabelle dette *log di snapshot* (descritte più oltre nel presente capitolo) per inviare le righe modificate dalla tabella master allo snapshot. Gli aggiornamenti rapidi sono disponibili solo per gli snapshot semplici. Gli aggiornamenti completi ricreano completamente lo snapshot. L'opzione force fa in modo che ORACLE utilizzi un aggiornamento rapido se questo è disponibile, altrimenti un aggiornamento completo.

La clausola start with specifica al database quando effettuare la prima replicazione 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 conservare il controllo sugli intervalli di replicazione è necessario specificare un valore per la clausola start with.

La clausola next specifica quanto a lungo ORACLE dovrà attendere tra un aggiornamento e il successivo. Tale clausola, dato che viene applicata a un differente tempo base ogni volta che lo snapshot è aggiornato, specifica un'espressione di data in luogo di una data fissa. Nell'esempio riportato in precedenza l'espressione della data è quella riportata di seguito.

```
next SysDate+7
```

Ogni volta che lo snapshot viene aggiornato, il successivo aggiornamento verrà programmato per sette giorni dopo. Anche se la programmazione dell'aggiornamento in questo esempio è piuttosto semplice, è possibile utilizzare numerose funzioni di ORACLE relative alla data per personalizzare le proprie pianificazioni di aggiornamento. Ad esempio, se si desiderasse effettuare l'aggiornamento ogni lunedì a mezzogiorno, a prescindere dalla data corrente, si potrebbe utilizzare la clausola next come nell'esempio seguente.

```
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 ulteriori informazioni sulle funzioni di ORACLE relative alla data si rimanda al Capitolo 8.

Per consentire l'aggiornamento automatico degli snapshot è necessario disporre di almeno un procedimento in background di aggiornamento di uno snapshot avviato nel proprio database. Il procedimento di aggiornamento, detto *SNPn* (dove *n* è un numero compreso tra 0 e 9) si attiva periodicamente e verifica se qualche snapshot nel database deve essere aggiornato.

Il numero di procedimenti *SNPn* avviati nel database è determinato da un parametro di inizializzazione detto *JOB_QUEUE_PROCESSES*. Tale parametro deve essere impostato, nel file *init.ora*, a un valore maggiore di 0. Nella maggioranza dei casi è sufficiente il valore 1.

L'intervallo in secondi tra le chiamate di attivazione ai procedimenti *SNPn* viene impostato per mezzo del parametro *JOB_QUEUE_INTERVAL* nel file *init.ora*. L'intervallo predefinito è pari a 60 secondi.

Se non si hanno procedimenti *SNPn* avviati, è necessario utilizzare metodi di aggiornamento manuale, descritti nel paragrafo seguente.

Aggiornamenti manuali

Oltre agli aggiornamenti automatici del database, è possibile effettuare aggiornamenti manuali dei propri snapshot, che prevalgono sugli aggiornamenti normalmente programmati. Il nuovo valore start with verrà basato sul momento dell'aggiornamento manuale.

Lo snapshot REGISTRO_LOCALE_TOTALI definito in precedenza conteneva, per gli intervalli di aggiornamento, le specifiche di seguito riportate.

```
refresh complete
start with SysDate next SysDate+7
```

Come riportato nel paragrafo precedente, le clausole start with e next forniscono i dati in input per la panificazione, da parte di ORACLE, degli aggiornamenti agli snapshot. È comunque possibile, in caso di significative modifiche alla tabella master, ridefinire tali impostazioni. Ad esempio, dopo un importante caricamento di dati nella tabella master, lo snapshot deve essere aggiornato per limitare le incoerenze tra lo snapshot stesso e la tabella master.

L'aggiornamento manuale degli snapshot è possibile per mezzo del package DBMS_SNAPSHOT messo a disposizione da ORACLE. Questo è un package pubblico di proprietà dell'utente SYS.

Per ulteriori informazioni su package e procedure si rimanda al Capitolo 24. All'interno di tale package è presente la procedura REFRESH che può essere utilizzata per l'aggiornamento di uno snapshot.

Di seguito è riportato un esempio di utilizzo del comando: l'utente esegue con execute della procedura, passando a questa due parametri. I parametri passati alla procedura sono descritti dopo l'esempio.

```
execute DBMS_SNAPSHOT.REFRESH('REGISTRO_LOCALE_TOTALI','?');
```

La procedura REFRESH del package DBMS_SNAPSHOT, visibile nell'esempio, utilizza due parametri. Il primo è il nome dello snapshot, che dovrebbe essere preceduto dal nome del proprietario dello snapshot stesso, se tale proprietario non è l'utente che esegue la procedura. Il secondo parametro è l'opzione di aggiornamento manuale. I valori disponibili per l'opzione di aggiornamento manuale sono riportati nella Tabella 28.1.

Tabella 28.1 Valori dell'opzione di aggiornamento manuale per la procedura DBMS_SNAPSHOT.REFRESH.

| OPZIONE DI AGGIORNAMENTO MANUALE | DESCRIZIONE |
|----------------------------------|--|
| F | Aggiornamento rapido (fast refresh). |
| f | Aggiornamento rapido (fast refresh). |
| C | Aggiornamento completo (complete refresh). |
| c | Aggiornamento completo (complete refresh). |
| ? | Indica che è necessario utilizzare per lo snapshot l'opzione predefinita di aggiornamento. |

Un'altra procedura del pacchetto DBMS_SNAPSHOT consente di aggiornare tutti gli snapshot programmati per l'aggiornamento automatico. Tale procedura, il cui nome è REFRESH_ALL, aggiornerà ciascuno snapshot separatamente. La procedura non accetta parametri. Ecco un esempio del suo utilizzo:

```
execute DBMS_SNAPSHOT.REFRESH_ALL;
```

Quando si esegue questo comando, ciascuno snapshot programmato per l'aggiornamento automatico viene aggiornato separatamente. Poiché gli snapshot non vengono aggiornati contemporaneamente, un'anomalia di funzionamento nel database o nel server durante l'esecuzione della procedura può causare la mancata sincronia degli snapshot locali con tutti gli altri. In tal caso sarà sufficiente avviare nuovamente la procedura, una volta recuperato il database.

NOTA *Su numerose piattaforme è disponibile l'utilità REFSNAP che esegue la procedura REFRESH_ALL. Per ulteriori informazioni si rimanda ai manuali di installazione e uso.*

Utilizzo dei gruppi di aggiornamento per gli snapshot

Gli snapshot correlati possono essere raccolti in *gruppi di aggiornamento*. Lo scopo di un gruppo di aggiornamento è quello di coordinare gli intervalli di aggiornamento dei propri membri. Gli snapshot le cui tabelle master hanno relazioni con le tabelle master di altri snapshot sono ottimi candidati alla qualità di membro in un gruppo di aggiornamento. La coordinazione degli intervalli di aggiornamento degli snapshot conserverà anche negli snapshot l'integrità referenziale delle tabelle master. Se non si utilizzano i gruppi di aggiornamento, i dati contenuti negli snapshot possono risultare incoerenti in relazione all'integrità referenziale delle tabelle master.

Tutte le modifiche ai gruppi di aggiornamento sono apportate per mezzo del package DBMS_REFRESH. Le procedure contenute in tale pacchetto sono MAKE, ADD, SUBTRACT, CHANGE, DESTROY e REFRESH. Nei paragrafi successivi viene descritto l'utilizzo di tali procedure per la gestione dei gruppi di aggiornamento. Le informazioni sui gruppi di aggiornamento esistenti possono essere ottenute per mezzo di query sulle viste dizionario del dati USER_REFRESH e USER_REFRESH_CHILDREN.

NOTA *Gli snapshot appartenenti a uno stesso gruppo di aggiornamento non devono obbligatoriamente appartenere allo stesso schema, ma devono essere memorizzati all'interno dello stesso database.*

Creazione di un gruppo di aggiornamento Per creare un gruppo di aggiornamento occorre eseguire la procedura MAKE del package DBMS_REFRESH. La struttura di tale procedura è riportata di seguito.

```
DBMS_REFRESH.MAKE
( name      IN VARCHAR2,
  list      IN VARCHAR2,
```

```

next_date IN DATE,
interval IN VARCHAR2,
implicit_destroy IN BOOLEAN DEFAULT FALSE,
lax IN BOOLEAN DEFAULT FALSE,
job IN BINARY_INTEGER DEFAULT 0,
rollback_seg IN VARCHAR2 DEFAULT NULL,
push_deferred_rpc IN BOOLEAN DEFAULT TRUE,
refresh_after_errors IN BOOLEAN DEFAULT FALSE );

```

Gli ultimi sei parametri della procedura hanno valori predefiniti normalmente accettabili. Per creare un gruppo di aggiornamento per gli snapshot REGISTRO creati in precedenza nel presente capitolo, si utilizza il comando riportato di seguito.

```

execute DBMS_REFRESH.MAKE
(name => 'Registro_group',
list => 'REGISTRO_LOCALE, REGISTRO_LOCALE_totali',
next_date => SysDate,
interval => SysDate+7)

```

NOTA Il parametro *list* dello snapshot, che è il secondo parametro nell'esempio, ha un apice singolo all'inizio e una alla fine, senza altri apici intermedi. Nell'esempio, i due snapshot REGISTRO_LOCALE e REGISTRO_LOCALE_TOTALI vengono passati alla procedura per mezzo di un unico parametro.

Il comando dell'esempio precedente crea un gruppo di aggiornamento di nome REGISTRO_GROUP, i cui membri sono i due snapshot basati su REGISTRO. Si noti che il nome del gruppo di aggiornamento è racchiuso tra apici singoli, come il parametro *list* che elenca i membri del gruppo (senza racchiudere tra apici ogni singolo membro).

Se il gruppo di aggiornamento si trova a contenere uno snapshot che è già membro di un altro gruppo di aggiornamento (ad esempio durante lo spostamento di uno snapshot da un gruppo di aggiornamento a un altro), è necessario impostare il parametro *lax* su TRUE. Uno snapshot può appartenere a un solo gruppo per volta.

Aggiunta di membri a un gruppo di aggiornamento Per aggiungere snapshot a un gruppo di aggiornamento esistente, si utilizza la procedura ADD del package DBMS_REFRESH. La struttura di tale procedura è riportata di seguito.

```

DBMS_REFRESH.ADD
( name      IN VARCHAR2,
list       IN VARCHAR2,
lax        IN BOOLEAN DEFAULT FALSE );

```

Come avviene per la procedura MAKE, il parametro *lax* della procedura ADD non deve essere specificato, a meno che uno snapshot non venga spostato da un gruppo di aggiornamento a un altro. Quando questa procedura viene eseguita con il parametro *lax* impostato su TRUE, lo snapshot viene spostato nel nuovo gruppo di aggiornamento e viene automaticamente eliminato dal vecchio gruppo di aggiornamento.

Rimozione di membri da un gruppo di aggiornamento Per rimuovere snapshot da un gruppo di aggiornamento esistente si utilizza la procedura SUBTRACT del package DBMS_REFRESH. La struttura di tale procedura è riportata di seguito.

```
DBMS_REFRESH.SUBTRACT
( name      IN VARCHAR2,
  list      IN VARCHAR2,
  lax      IN BOOLEAN DEFAULT FALSE );
```

Come avviene per le procedure MAKE e ADD, come dati in input per questa procedura è possibile utilizzare un singolo snapshot o un elenco di snapshot (separati da virgole).

Modifica dell'intervallo di aggiornamento di un gruppo di aggiornamento Per modificare l'intervallo di aggiornamento di un gruppo si utilizza la procedura CHANGE del package DBMS_REFRESH. La struttura di tale procedura è riportata di seguito.

```
DBMS_REFRESH.CHANGE
( name      IN VARCHAR2,
  next_date   IN DATE DEFAULT NULL,
  interval    IN VARCHAR2 DEFAULT NULL,
  implicit_destroy IN BOOLEAN DEFAULT NULL,
  rollback_seg   IN VARCHAR2 DEFAULT NULL,
  push_deferred_rpc IN BOOLEAN DEFAULT NULL,
  refresh_after_errors IN BOOLEAN DEFAULT NULL);
```

Il parametro *next_date* è analogo alla clausola start with nel comando create snapshot. Il parametro *interval* è analogo alla clausola next nel comando create snapshot.

Ad esempio, per modificare l'intervallo di aggiornamento del gruppo REGISTRO_GROUP in modo che il gruppo venga replicato ogni tre giorni è possibile eseguire il comando riportato di seguito, che specifica un valore NULL per il parametro *next_date*, lasciando tale valore invariato.

```
execute DBMS_REFRESH.CHANGE
(name => 'Registro_group',
 next_date => null,
 interval => SysDate+3);
```

Completato questo comando, il ciclo di aggiornamento per il gruppo REGISTRO_GROUP verrà modificato in modo da aggiornare il gruppo ogni tre giorni.

Eliminazione di un gruppo di aggiornamento Per eliminare un gruppo di aggiornamento si utilizza la procedura DESTROY del package DBMS_REFRESH. La struttura di tale procedura è riportata di seguito. L'unico parametro della procedura è il nome del gruppo di aggiornamento.

```
execute DBMS_REFRESH.DESTROY(name => 'Registro_group');
```

È inoltre possibile impostare l'eliminazione implicita del gruppo di aggiornamento. Se al momento della creazione del gruppo per mezzo della procedura MAKE si imposta il parametro *implicit_destroy* su TRUE, il gruppo di aggiornamento verrà eliminato non appena sarà stato rimosso dal gruppo stesso l'ultimo membro (normalmente per mezzo della procedura SUBTRACT).

Aggiornamento manuale di un gruppo di aggiornamento Per aggiornare manualmente un gruppo di aggiornamento si utilizza la procedura REFRESH del package DBMS_REFRESH. La procedura REFRESH accetta come unico parametro il nome del gruppo di aggiornamento. Il comando riportato di seguito aggiorna il gruppo di aggiornamento di nome REGISTRO_GROUP.

```
execute DBMS_REFRESH.REFRESH(name => 'Registro_group');
```

28.8 Snapshot e trigger

Invece di utilizzare gli snapshot per replicare i dati, è possibile utilizzare i trigger per replicare i dati transazione per transazione, su copie remote di una tabella (Capitolo 23). Per scegliere se utilizzare i trigger o gli snapshot per le proprie esigenze, è necessario tenere in considerazione i punti elencati di seguito a favore degli snapshot.

1. I trigger di replicazione (come quelli utilizzati per la revisione delle applicazioni) vengono eseguiti ogni volta che viene modificata una riga della tabella master. Se la tabella è sottoposta a numerose transazioni, questa circostanza può limitare fortemente le prestazioni dell'applicazione e della rete. Gli snapshot riuniscono tutte le modifiche in un'unica (ma di dimensioni potenzialmente rilevanti) transazione.
2. Le transazioni mediante trigger non possono essere programmate, in quanto intervengono quando interviene la transazione nella tabella master. Le replicazioni per mezzo di snapshot possono essere programmate o eseguite manualmente. Ambedue i metodi consentono di evitare grossi carichi per la replicazione delle transazioni durante i momenti di picco nell'utilizzo dell'applicazione.
3. Quando cambia la complessità delle transazioni nella tabella master, anche i trigger devono cambiare. Gli snapshot, le cui relazioni con le tabelle base sono dichiarate, non devono cambiare.
4. Se vengono replicate più tabelle correlate, la sincronizzazione delle replicazioni può essere ottenuta per mezzo di gruppi di aggiornamento degli snapshot. La conservazione di repliche di tabelle correlate per mezzo di trigger richiede rilevanti sforzi di programmazione.
5. Se la base per la replicazione è una query complessa (come un'operazione di unione, group by o connect by), normalmente non si desidera ripetere la valutazione della query per ciascuna transazione nella tabella master. Gli snapshot possono agevolare la programmazione della replicazione di query complesse. Esiste un solo campo in cui i trigger sono superiori agli snapshot per quanto riguarda la replicazione. Se i dati nella tabella replicata devono essere costantemente sincro-

nizzati con la tabella master, è necessario utilizzare i trigger o intervalli di aggiornamento molto brevi per gli snapshot. Se la tabella master ha un trigger su di sé e la transazione sulla copia remota fallisce, la transazione sulla tabella master fallirà anch'essa, conservando la coerenza tra le due tabelle.

Nella replicazione, come nell'integrità referenziale, è necessario utilizzare un metodo dichiarativo (snapshot) in luogo di un metodo procedurale (trigger), a meno che il metodo dichiarativo non sia inadatto per le proprie esigenze.

28.9 Sintassi per il comando `create snapshot log`

Negli snapshot semplici ciascun record è basato su una singola riga di una singola tabella master. Quando si utilizzano snapshot semplici, è possibile creare un *log di snapshot* sulla tabella master. Il log di snapshot è una tabella che registra la data in cui è stata replicata per l'ultima volta ciascuna riga modificata nella tabella master. La registrazione delle righe modificate può essere poi utilizzata durante gli aggiornamenti per inviare agli snapshot solo le righe della tabella master che sono state modificate. Più snapshot semplici basati sulla stessa tabella possono utilizzare lo stesso log di snapshot. La sintassi per il comando `create snapshot log` è riportata di seguito. Il comando prevede tutti i parametri normalmente associati con le tabelle.

```
create snapshot log on [schema.]tabella
  [ with {[primary key] [,rowid] [,,(colonna filtro)
            | ,(colonna filtro) ...]}
    [ pctfree intero
    | pctused intero
    | initrans intero
    | maxtrans intero
    | tablespace tablespace
    | storage memorizzazione];
```

Il comando `create snapshot log` viene eseguito nel database della tabella master, normalmente dal proprietario della tabella master stessa. I log di snapshot non dovranno essere creati per tabelle coinvolte solo in snapshot complessi, in quanto non verrebbero utilizzati. Per il log di snapshot non viene specificato alcun nome.

Per creare un log di snapshot per la tabella REGISTRO, occorre eseguire il comando riportato di seguito, dall'interno dell'account proprietario della tabella.

```
create snapshot log on REGISTRO
with ROWID
tablespace SNAP_LOGS
storage (initial 40K next 40K pctincrease 0);
```

Il comando riportato nell'esempio precedente crea un log di snapshot in un tablespace di nome SNAP_LOGS. Poiché la crescita dei log di snapshot nel tempo non è prevedibile, è possibile memorizzare gli oggetti a essi associati in tablespace appositi. Poiché lo snapshot REGISTRO_LOCALE è stato creato utilizzando la clausola `with ROWID`, il log di snapshot per la tabella REGISTRO dovrebbe essere creato utilizzando la clausola `with ROWID`.

Privilegi di sistema richiesti

Per creare log di snapshot è necessario disporre dei privilegi di sistema CREATE TABLE e CREATE TRIGGER. Se si crea il log di snapshot da un account utente che non possiede la tabella master, è necessario disporre dei privilegi di sistema CREATE ANY TABLE e CREATE ANY TRIGGER.

Oggetti locali e remoti

Quando si crea uno snapshot, vengono creati due oggetti nello schema della tabella master. Dal punto di vista del proprietario dello snapshot, il comando create snapshot log viene eseguito nel database remoto e gli oggetti da tale comando creati si trovano tutti nel database remoto. Il log di snapshot crea una tabella e un trigger.

- Viene creata una tabella per memorizzare i valori ROWID delle righe della tabella master che vengono modificate (se viene creato uno snapshot basato sui valori ROWID), insieme a una colonna separata che registra il momento in cui sono state replicate per l'ultima volta le righe modificate. Se si crea uno snapshot utilizzando la clausola with primary key, il log di snapshot conterrà colonne relative alle colonne di chiave primaria della tabella master. Questa tabella si chiama MLOG\$_nome_tabella_master. Se viene creato un log di snapshot sulla tabella REGISTRO, la tabella log di snapshot verrà chiamata MLOG\$_REGISTRO.
- Viene creato un trigger AFTER ROW per l'inserimento dei valori chiave, siano essi valori ROWID o valori chiave primaria, e date di aggiornamento delle righe aggiornate, inserite o eliminate nella tabella log di snapshot. Tale trigger prende il nome TLOG\$_nome_tabella_master. Se il log di snapshot viene creato sulla tabella REGISTRO, il trigger prenderà il nome di TLOG\$_REGISTRO.

Le colonne “filtro” mostrate nel comando create snapshot log sono correlate all'utilizzo di sottoquery all'interno di query su snapshot. L'utilizzo di sottoquery all'interno di query su snapshot è possibile in ORACLE8.0, ma solo se la sottoquery conserva completamente i valori chiave dei dati replicati. Le sottoquery possono essere utilizzate solo con snapshot basati su valori di chiave primaria.

A causa dei nomi utilizzati per gli oggetti creati per i log di snapshot, il nome della tabella su cui il log di snapshot è basato non può superare la lunghezza di 19 caratteri.

28.10 Visualizzazione di informazioni su snapshot esistenti

Per visualizzare dati riguardanti gli snapshot, occorre effettuare una query sulla vista del dizionario di dati USER_SNAPSHOTS. Per informazioni sugli snapshot a cui l'utente è in grado di accedere, ma di cui non è proprietario, occorre effettuare una query su ALL_SNAPSHOTS. Nell'esempio seguente è riportata una query su USER_SNAPSHOTS.

```

select
  Name,          /*Nome della vista utilizzata per lo snapshot*/
  Last_Refresh, /*Data dell'ultimo aggiornamento */
  Type,          /*Tipo di aggiornamento utilizzato per aggiornamenti
automatici */
  Query         /*Query utilizzata per creare lo snapshot*/
  from USER_SNAPSHOTS;

```

Questa query restituisce le informazioni più utili riguardanti lo snapshot. Nella Tabella 28.2 è riportato l'elenco completo delle colonne disponibili.

Tabella 28.2 Colonne di USER_SNAPSHOTS.

| NOME COLONNA | DESCRIZIONE |
|---------------------|---|
| Owner | Proprietario dello snapshot. |
| Name | Nome dello snapshot. |
| Table_Name | Nome della tabella base locale per lo snapshot (SNAP\$_nomesnapshot). |
| Master_View | Nome della vista locale utilizzata per gli aggiornamenti (MVIEW\$_nomesnapshot). |
| Master_Owner | Proprietario della tabella master. |
| Master | Nome della tabella master. |
| Master_Link | Nome del link di database utilizzato dalla query dello snapshot. |
| Can_Use_Log | Flag che indica se lo snapshot è in grado di utilizzare un log di snapshot (i valori sono YES e NO). |
| Updatable | Flag che indica se lo snapshot può essere aggiornato. |
| Refresh_Method | Valori utilizzati per pilotare l'aggiornamento rapido di uno snapshot. |
| Last_Refresh | Data e ora presso il sito master dell'ultimo aggiornamento dello snapshot. |
| Error | Qualsiasi errore restituito durante l'ultimo tentativo di aggiornamento. |
| FR_Operations | Stato delle operazioni di aggiornamento rapido ('REGENERATE' o 'VALID'). |
| CR_Operations | Stato delle operazioni di aggiornamento completo ('REGENERATE' o 'VALID'). |
| Type | Tipo di aggiornamento da utilizzare durante gli aggiornamenti automatici (FAST, COMPLETE o FORCE). |
| Next | Funzione data utilizzata per calcolare la data e l'ora del prossimo aggiornamento. |
| Start_With | Funzione data utilizzata per calcolare la data e l'ora del primo aggiornamento. |
| Refresh_Group | Nome del gruppo di aggiornamento a cui lo snapshot appartiene. |
| Update_Trig | NULL; presente per compatibilità con le versioni precedenti di ORACLE. |
| Update_Log | Nome della tabella su cui vengono registrate le modifiche (per gli snapshot aggiornabili). |
| Query | Query che forma la base dello snapshot. |
| Master_Rollback_Seg | Segmento di rollback utilizzato durante il riempimento e le operazioni di aggiornamento degli snapshot. |

Informazioni sui log di snapshot possono essere ottenute per mezzo di query sulla vista del dizionario di dati **USER_SNAPSHOT_LOGS**, le cui colonne sono elencate nella Tabella 28.3.

Tabella 28.3 Colonne di **USER_SNAPSHOT_LOGS**.

| NOME COLONNA | DESCRIZIONE |
|-------------------|---|
| Log_Owner | Proprietario del log di snapshot. |
| Master | Nome della tabella master a cui il log di snapshot log si applica. |
| Log_Table | Nome della tabella di log di snapshot (MLOG\$_nome_tabella_master). |
| Log_Trigger | Nome del trigger sulla tabella master che popola la tabella log di snapshot (TLOG\$_nome_tabella_master). |
| ROWIDs | Flag che indica se lo snapshot è basato su ROWID. |
| Primary_Keys | Flag che indica se lo snapshot è basato su chiavi primarie. |
| Filter_Columns | Colonne filtro, se una sottoquery fa parte della query base dello snapshot. |
| Current_Snapshots | Data in cui lo snapshot della tabella master è stato aggiornato per l'ultima volta. |
| Snapshot_ID | Numero unico d'identificazione per lo snapshot. |

È consigliabile effettuare query periodiche su tali viste del dizionario di dati, per determinare se qualche snapshot non è stato aggiornato per lungo tempo. Lunghi intervalli tra gli aggiornamenti possono causare problemi di sincronia e una crescita a dimensioni troppo ampie dei log di snapshot.

28.11 Modifica di snapshot e log di snapshot

Per gli snapshot esistenti è possibile modificare i parametri di memorizzazione, le opzioni di aggiornamento e gli intervalli di aggiornamento. Se non si è sicuri delle impostazioni correnti per uno snapshot, occorre controllare la vista del dizionario di dati **USER_SNAPSHOTS** come descritto nel paragrafo precedente.

Di seguito è riportata la sintassi per il comando **alter snapshot**.

```
alter snapshot [utente.]snapshot
  [ pctfree intero
  | pctused intero
  | initrans intero
  | maxtrans intero
  | storage memorizzazione ] ...
[using {index [pctfree intero
  | pctused intero
  | initrans intero
  | maxtrans intero
```

```

| storage storage] ...
|[ default] master rollback segment [segmento_rollback]}
[refresh [fast | complete | force]
[start with data] [next data] [with primary key ]

```

Ad esempio, per modificare l'opzione di aggiornamento utilizzata per lo snapshot REGISTRO_LOCALE si esegue il comando riportato di seguito.

```
alter snapshot REGISTRO_LOCALE
refresh complete;
```

Tutti i successivi aggiornamenti dello snapshot REGISTRO_LOCALE aggiorneranno l'intera tabella base locale dello snapshot stesso.

Per modificare uno snapshot è necessario essere proprietari dello stesso o disporre del privilegio di sistema ALTER ANY SNAPSHOT.

I parametri di memorizzazione per i log di snapshot possono essere modificati per mezzo del comando alter snapshot log. Di seguito ne è riportata la sintassi.

```
alter snapshot log on [utente.]tabella
[ add {[primary key] [,rowid] [,,(colonna_filtro)
| ,,(colonna_filtro)] ...}
[ pctfree intero
| pctused intero
| initrans intero
| maxtrans intero
| storage memorizzazione] ;
```

La modifica dei parametri correlati alla memorizzazione causa la modifica dei parametri di storage sulla tabella log di snapshot. Ad esempio, il comando descritto di seguito modifica il parametro next all'interno della clausola storage per il log di snapshot.

```
alter snapshot log on REGISTRO
storage (next 100K);
```

Per modificare un log di snapshot è necessario essere proprietari della tabella, disporre del privilegio ALTER per la tabella stessa o disporre del privilegio di sistema ALTER ANY TABLE.

28.12 Scaricamento di snapshot e log di snapshot

Per scaricare uno snapshot è necessario disporre dei privilegi di sistema richiesti per eliminare sia lo snapshot stesso, sia tutti gli oggetti a questo correlati. Serve il privilegio di sistema DROP SNAPSHOT o DROP ANY SNAPSHOT se lo snapshot non si trova nel proprio schema.

Di seguito è riportato il comando per scaricare lo snapshot REGISTRO_LOCALE_TOTALI creato in precedenza nel presente capitolo.

```
drop snapshot REGISTRO_LOCALE_TOTALS;
```

I log di snapshot possono essere scaricati per mezzo del comando `drop snapshot log`. Una volta scaricato il log di snapshot da una tabella master, non saranno possibili aggiornamenti rapidi per gli snapshot semplici basati su tale tabella. Un log di snapshot dovrebbe essere eliminato quando non esistono snapshot semplici basati sulla tabella master. Nell'esempio seguente è riportato il comando per l'eliminazione del log di snapshot creato sulla tabella REGISTRO, descritto in precedenza nel presente capitolo.

```
drop snapshot log on REGISTRO;
```

Per eliminare un log di snapshot è necessario essere in grado di eliminare sia il log stesso, sia gli oggetti a questo correlati. Se si è proprietari del log di snapshot è necessario disporre dei privilegi di sistema `DROP TABLE` e `DROP TRIGGER`. Se non si è proprietari del log di snapshot è necessario disporre dei privilegi di sistema `DROP ANY TABLE` e `DROP ANY TRIGGER`.

• Capitolo 29

• Utilizzo di ConText per ricerche di testo

- 29.1 Aggiunta di testo al database
- 29.2 Query di testo dal database
- 29.3 Impostazione dell'opzione ConText

Quando l'archivio è in funzione da un po' di tempo, Talbot nota una tendenza comune a numerose applicazioni di database: la quantità di testo memorizzata cresce. Nuovi candidati inviano i loro curriculum e questi vengono aggiunti alla tabella PROSPETTIVA. I dipendenti vengono valutati e le valutazioni sono aggiunte al database.

Con il crescere della quantità di testo nel database, cresce anche la complessità delle query di testo effettuate verso il database stesso. Invece di eseguire solamente ricerche di stringhe, è ora necessario utilizzare nuove funzioni di ricerca, come l'attribuzione di priorità durante ricerche di più termini o l'ordinamento in graduatoria dei risultati di una ricerca di testo.

A partire da ORACLE7.3 è possibile utilizzare l'opzione ConText del server ORACLE per effettuare ricerche basate sul testo. ConText può essere utilizzata per effettuare ricerche con caratteri jolly, ordinamenti per rilevanza basati su logica "fuzzy", ricerche di prossimità, ponderazione dei termini ed espansione delle parole. Nel presente capitolo viene descritto l'utilizzo di ConText come parte di una query che comprenda la ricerca di testo. Nel capitolo seguente viene descritta nei dettagli l'impostazione del database e delle tabelle necessaria per consentire l'esecuzione di query con ConText.

29.1 Aggiunta di testo al database

Il tasto può essere aggiunto al database memorizzando fisicamente il testo stesso in una tabella o memorizzando nel database puntatori a file esterni. Per quanto riguarda i candidati di Talbot è possibile memorizzare i curriculum nel database o in file esterni; nel secondo caso sarà necessario memorizzare nel database i nomi dei file stessi.

Per memorizzare i curriculum nel database è possibile espandere la tabella PROSPETTIVA. In origine questa tabella comprendeva solo due colonne:

```
create table PROSPETTIVA (
  Nome        VARCHAR2(25) not null,
```

```
Indirizzo      VARCHAR2(35)
)
/
```

Per supportare i dati di testo viene aggiunta alla tabella PROSPETTIVA una terza colonna:

```
alter table PROSPETTIVA add
(Resume      LONG)
/
```

Alla colonna Resume della tabella PROSPETTIVA viene assegnato il tipo di dati LONG. Nell'esempio seguente viene aggiunto un singolo valore Resume alla tabella PROSPETTIVA. La voce di PROSPETTIVA per Wilfred Lowell viene aggiornata. Nel listato viene immesso un ritorno a capo dopo ciascuna riga.

```
update PROSPETTIVA
   set Resume = 'Esperienza con zappa, forcione e vari attrezzi da
giardino.
Ottime referenze dal precedente datore di lavoro signor Garden.
Non esagera con le forbici. Socializza con i compagni.
Tra i titoli di studio, una laurea in ingegneria.'
   where Nome = 'WILFRED LOWELL';
```

Una volta aggiornata la voce di Wilfred Lowell in modo che essa comprenda un valore Resume, è possibile selezionare tale valore dal database.

```
select Resume
  from PROSPETTIVA
 where Nome = 'WILFRED LOWELL';
```

Di seguito è riportato il risultato della query precedente.

RESUME

```
-----
Esperienza con zappa, forcione e vari attrezzi da giardino.
Ottime refer
```

Per default, solo i primi 80 caratteri dei tipi di dati LONG vengono visualizzati. Per modificare questa impostazione occorre utilizzare il comando set long in SQL-PLUS, come nell'esempio seguente.

```
set long 1000
```

```
select Resume
  from PROSPETTIVA
 where Nome = 'WILFRED LOWELL';
```

RESUME

```
-----
Esperienza con zappa, forcione e vari attrezzi da giardino.
Ottime referenze dal precedente datore di lavoro signor Garden.
Non esagera con le forbici. Socializza con i compagni.
Tra i titoli di studio, una laurea in ingegneria.
```

Il testo memorizzato nelle colonne LONG comprende i ritorni a capo immessi al momento di specificare il valore Resume.

29.2 Query di testo dal database

Si supponga di voler visualizzare i nomi di tutti i candidati che hanno esperienza o laurea in ingegneria. La query potrebbe assumere la forma riportata di seguito.

```
select Nome
  from PROSPETTIVA
 where Resume like '%ingegneria%';
```

Tale query però fallirebbe, in quanto Resume è impostato come tipo di dati LONG. Di seguito è riportato il messaggio d'errore.

```
where resume like '%ingegneria%'
*
ERROR at line 2:
ORA-00932: inconsistent datatypes
```

Come si fa a determinare la lunghezza del Resume più lungo, per utilizzare tale dato in un comando set long?

```
select LENGTH(Resume)
  from PROSPETTIVA
 where Nome = 'WILFRED LOWELL';

select LENGTH(Resume)
*
ERROR at line 1:
ORA-00932: inconsistent datatypes
```

L'errore “inconsistent datatypes” si verifica in quanto si sta tentando di eseguire funzioni di manipolazione delle stringhe di testo su una colonna (Resume) che ha un tipo di dati LONG. Se si fosse utilizzato invece un tipo di dati VARCHAR2, le query descritte in precedenza sarebbero state completate con successo. Il tipo di dati VARCHAR2 avrebbe però limitato la memorizzazione del testo. In ORACLE7 i tipi di dati VARCHAR2 sono in grado di memorizzare solo 2000 caratteri; in ORACLE8 4000.

Per questo motivo, se si desidera memorizzare più di 4000 caratteri di testo per colonna, non è possibile utilizzare un tipo di dati VARCHAR2 per le ricerche di testo. Se si fossero utilizzati tipi di dati VARCHAR2 per i campi di testo, si sarebbe dovuto decidere come memorizzare i dati: memorizzare il testo in maiuscolo o forzare gli utenti a utilizzare funzioni per le stringhe di testo come UPPER al momento di effettuare le proprie ricerche?

Inoltre, se il testo è già formattato, ad esempio in un documento creato utilizzando Microsoft Word, è possibile utilizzare il tipo di dati LONG RAW per memorizzare il file all'interno del database. Come si fa a eseguire query su un documento di questo tipo utilizzando le funzioni di ricerca testuale disponibili in ORACLE?

Query ConText

A partire da ORACLE7.3 è possibile utilizzare l'opzione ConText per effettuare ricerche di testo come parte di query sul database. Le ricerche di testo sono implementate in ORACLE tramite processi server che operano in parallelo ai normali processi in background. L'impostazione dei processi server ConText è descritta nel Capitolo 30.

Una volta preparato il database, 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 raccolta di tabelle e indici che memorizzano informazioni sul testo a sua volta memorizzato nella colonna. L'impostazione degli indici testuali è descritta nel Capitolo 30.

Se il database è impostato in modo da utilizzare l'opzione ConText e un indice testuale viene creato sulla colonna Resume della tabella PROSPETTIVA, le capacità di ricerca testuale crescono enormemente. È ora possibile cercare qualsiasi curriculum che contenga la parola ‘ingegneria’.

```
select Nome  
      from PROSPETTIVA  
     where CONTAINS(Resume, 'ingegneria')>0;
```

La funzione CONTAINS richiede al processo server ConText la verifica dell'indice testuale per la colonna Resume. Se nell'indice testuale della colonna Resume viene individuata la parola ‘ingegneria’, viene restituito un *punteggio* maggiore di zero e il valore Nome corrispondente. Il punteggio rappresenta una valutazione della corrispondenza del record restituito con i criteri specificati nella funzione CONTAINS.

Funzionamento di una query ConText Quando in una query si utilizza una funzione ConText come CONTAINS, la parte testuale della query viene elaborata dai processi server ConText. Il resto della query viene elaborato esattamente come una normale query all'interno del database. I risultati dell'elaborazione ConText e dell'elaborazione “normale” della query vengono raccolti, in modo da restituire una singola serie di record all'utente.

Espressioni di query ConText disponibili

L'opzione ConText sarebbe piuttosto limitata se consentisse esclusivamente la ricerca di parole che coincidano con esattezza. In realtà ConText mette a disposizione un'ampia varietà di funzioni per la ricerca del testo, che è possibile utilizzare per personalizzare le proprie query. La maggior parte delle funzioni di ricerca del testo viene attivata per mezzo della funzione CONTAINS, che può 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 l'esecuzione delle ricerche di testo elencate di seguito.

- Esatta coincidenza di una parola o frase.
- Esatta coincidenza di più parole, utilizzando la logica booleana per combinare le ricerche.

- Ricerche basate sulla prossimità delle parole alle altre parole nel testo.
- Ricerche di parole che condividono la stessa etimologia.
- Coincidenza “fuzzy” delle parole.
- Ricerche di parole che abbiano lo stesso suono di altre.

Nei paragrafi seguenti sono riportati esempi di ciascuno di questi tipi di ricerca testuale, insieme ad informazioni sugli operatori che è possibile utilizzare per personalizzare le proprie ricerche.

Ricerca dell'esatta coincidenza di una parola

Nella tabella PROSPETTIVA un singolo valore Resume è stato aggiornato in modo da includere informazioni su Wilfred Lowell. Con il crescere del database, cresce anche il numero dei curriculum. Nell'esempio seguente è riportato il primo curriculum immesso nella tabella PROSPETTIVA. Le query riportate nel resto del capitolo utilizzeranno metodi di ricerca differenti per il recupero di questi dati.

```
set long 1000

select Resume
  from PROSPETTIVA
 where Nome = 'WILFRED LOWELL';

RESUME
-----
Esperienza con zappa, forcione e vari attrezzi da giardino.
Ottime referenze dal precedente datore di lavoro signor Garden.
Non esagera con le forbici. Socializza con i compagni.
Tra i titoli di studio, una laurea in ingegneria.
```

Nell'esempio seguente viene effettuata una query sulla tabella PROSPETTIVA per tutti i candidati il cui nome contenga la parola ‘ingegneria’:

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegneria')>0;
```

All'interno della funzione CONTAINS, il simbolo ‘>’ viene detto *operatore di soglia*. La precedente ricerca di testo può essere tradotta in italiano come segue:

seleziona tutti i valori della colonna Nome
dalla tabella PROSPETTIVA
dove il punteggio della ricerca di testo sulla colonna Resume
per una corrispondenza esatta con la parola 'ingegneria'
superà un valore soglia di 0.

L'analisi di soglia confronta il punteggio, cioè il punteggio interno calcolato da ConText 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 ricorrenza della stringa di testo all'interno del testo esaminato.

È anche possibile visualizzare il punteggio come parte della query, tramite la funzione SCORE. La funzione SCORE prevede un singolo parametro: un'etichetta da assegnare al punteggio all'interno della ricerca di testo.

```
select Nome, SCORE(10)
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegneria', 10)>0;
```

In questo esempio 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.

La funzione SCORE può essere utilizzata in select (come nella query precedente), in una clausola group by o in una clausola order by.

Ricerca dell'esatta coincidenza di più parole

Per combinare i risultati di più ricerche di testo in una singola query è possibile utilizzare la logica booleana (operatori AND e OR). È inoltre possibile cercare più termini all'interno della stessa funzione CONTAINS e fare in modo che ConText risolva i risultati della ricerca. Se ad esempio si desidera cercare i candidati il cui testo di curriculum contenga le parole 'ingegneria' e 'meccanica', è possibile utilizzare la query descritta di seguito.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegneria & meccanica')>0;
```

Il carattere '&' è il simbolo utilizzato per l'operazione AND: la funzione CONTAINS restituisce una riga solo se Resume contiene ambedue le parole 'ingegneria' e 'meccanica'. Ciascuna ricerca dovrà superare il criterio di soglia definito per i punteggi di ricerca. Il simbolo '&' può essere sostituito con la parola AND, come nell'esempio seguente.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegneria AND meccanica')>0;
```

La query in questo caso non ricerca la frase 'ingegneria and meccanica', in quanto la parola AND viene utilizzata per combinare le due condizioni di ricerca. Nel paragrafo seguente viene descritta la ricerca di frasi.

Per cercare più di due termini, basta semplicemente aggiungere i termini cercati alla clausola CONTAINS, come nell'esempio seguente.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegneria & meccanica & turbina')
       >0;
```

Questa query restituisce una riga solo se i punteggi di ricerca per le parole 'ingegneria', 'meccanica' e 'turbina' sono tutti maggiori di zero.

Oltre all'operatore AND è possibile utilizzare l'operatore OR. In tal caso viene restituito un record se una delle condizioni di ricerca soddisfa la soglia definita. Il simbolo per l'operatore OR nelle ricerche ConText è una barra verticale (|). Le due query seguenti sono pertanto equivalenti.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegneria OR meccanica')>0;

select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegneria | meccanica')>0;
```

Quando vengono eseguite le query precedenti, viene restituito un record solo se una delle due ricerche separate (di 'ingegneria' e 'meccanica') restituisce un punteggio maggiore di zero.

L'operatore ACCUM ("accumula") mette a disposizione un ulteriore 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.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegneria ACCUM meccanica')>0;

select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegneria, meccanica')>0;
```

È inoltre possibile utilizzare ConText per sottrarre i punteggi di più ricerche prima di confrontare il risultato con il punteggio di soglia. L'operatore MINUS sottrae il punteggio della ricerca del secondo termine dal punteggio della ricerca del primo. La query nell'esempio seguente determina il punteggio di ricerca per 'ingegneria' e lo sottrae dal punteggio di ricerca per 'meccanica'. La differenza viene confrontata con il punteggio di soglia.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegneria MINUS meccanica')>0;
```

In luogo dell'operatore MINUS è possibile utilizzare il simbolo '-', come nell'esempio seguente.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegneria - meccanica')>0;
```

Per rendere più chiara la logica dei criteri di ricerca è possibile utilizzare le parentesi. Ad esempio, se il criterio di ricerca contiene sia operatori AND, sia operatori OR, è possibile utilizzare le parentesi per rendere chiaro il modo in cui le righe vengono esaminate.

Ad esempio, la query seguente restituisce una riga se il valore esaminato contiene la parola ‘scavare’ o contiene ambedue le parole ‘zappare’ e ‘piantare’.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'scavare OR (zappare AND piantare)')
   >0;
```

Modificando la posizione delle parentesi si modifica la logica della funzione CONTAINS. La query seguente restituisce una riga se il testo esaminato contiene la parola ‘scavare’ o la parola ‘zappare’ e contiene anche la parola ‘piantare’.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, '(scavare OR zappare) AND piantare')
   >0;
```

Quando si valutano i punteggi di ricerche multiple è possibile fare in modo che ConText valuti i punteggi di alcune ricerche come più importanti di altri. Ad esempio, se si desidera che il punteggio di ‘ingegneria’ valga il doppio quando confrontato con il punteggio di soglia, è possibile utilizzare il simbolo ‘*’ per indicare il fattore per cui il punteggio di ricerca deve essere moltiplicato.

La query seguente raddoppia il punteggio di ricerca per ‘ingegneria’ nella valutazione all’interno di una condizione OR.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegneria*2 OR meccanica*1')>5;
```

L’attribuzione di un “peso” ai punteggi delle ricerche può essere utilizzata allo scopo di indicare l’importanza dei termini nella ricerca condotta. Se un termine è il più importante nella ricerca, è possibile assegnare a tale termine il “peso” maggiore. Per mezzo degli operatori AND, OR, ACCUM e MINUS è possibile ricercare qualsiasi combinazione di coincidenze dei termini. Nel paragrafo seguente è descritta la ricerca di frasi.

Ricerca dell’esatta coincidenza di una frase

Quando si cerca l’esatta coincidenza di una frase, è necessario specificare l’intera frase come parte della funzione CONTAINS. Se la frase comprende parole riservate (come ‘and’, ‘or’, ‘minus’), sarà necessario utilizzare gli operatori di raggruppamento descritti nel presente paragrafo, in modo che la ricerca venga effettuata correttamente.

Nella query seguente si cercano tutti coloro la cui colonna Resume contenga la frase ‘ottime referenze’. La riga di Wilfred Lowell verrà restituita anche se la parola ‘ottime’ ha l’iniziale maiuscola (‘Ottime’).

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ottime referenze')>0;
```

Se la frase cercata contiene una parola riservata in ConText, è necessario utilizzare parentesi graffe ({}) per racchiudere il testo. La query seguente cerca la frase ‘giorno e notte’. La parola ‘and’ è racchiusa tra parentesi graffe.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'giorno {and} notte')>0;
```

La ricerca di ‘giorno {and} notte’ è diversa dalla ricerca di ‘giorno and notte’. La query di ‘giorno {and} notte’ restituisce un record solo se nel testo esaminato esiste la frase ‘giorno and notte’. La query di ‘giorno and notte’ restituisce un record se il punteggio di ricerca per la parola ‘giorno’ e il punteggio di ricerca per la parola ‘notte’ sono ambedue superiori al punteggio di soglia.

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. La query seguente cerca la frase ‘giorno and notte’ racchiudendola interamente tra parentesi graffe.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, '{giorno and notte}')>0;
```

Ricerca di parole vicine le une alle altre

La funzione di *ricerca di prossimità* di ConText consente di effettuare ricerche di testo basate sulla vicinanza dei termini cercati nel documento esaminato. Una ricerca di prossimità restituisce un punteggio alto per parole adiacenti e un punteggio basso per parole separate nettamente tra loro. Se le parole sono immediatamente adiacenti la ricerca di prossimità restituisce un punteggio pari a 100.

Per effettuare una ricerca di prossimità si utilizza la parola chiave NEAR, come nell’esempio seguente.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'studio NEAR ingegneria')>0;
```

L’operatore NEAR può essere sostituito con il suo simbolo equivalente, il punto e virgola (;). Di seguito è riportata la query modificata.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'studio ; ingegneria')>0;
```

I metodi di ricerca di frasi e parole descritti nel presente capitolo possono essere utilizzati per cercare esatte coincidenze di parole e frasi e per effettuare ricerche di prossimità. Sino a qui, tutte le ricerche utilizzano esatte coincidenze dei termini cercati come base. Nei paragrafi seguenti viene descritta l’espansione dei termini cercati per mezzo di quattro metodi: i caratteri jolly, la radice delle parole, la coincidenza “fuzzy” e le ricerche SOUNDEX.

Utilizzo di caratteri jolly durante le ricerche

Nei precedenti esempi del presente capitolo sono stati esaminati i curriculum di PROSPETTIVA alla ricerca di valori di testo che coincidessero esattamente con i criteri specificati. Ad esempio, si è cercata la parola ‘ingegneria’ ma non la parola ‘ingegnere’. I caratteri jolly consentono di espandere l’elenco dei termini validi per la ricerca da utilizzarsi nella query.

Come nella normale elaborazione delle stringhe di testo, per la ricerca sono disponibili due caratteri jolly.

| <i>Carattere</i> | <i>Descrizione</i> |
|------------------|---|
| % | Simbolo di percentuale: carattere jolly per caratteri multipli |
| _ | Trattino di sottolineatura: carattere jolly per singolo carattere |

La seguente query ConText cerca tutte le coincidenze di testo per tutte le parole che iniziano con i caratteri ‘ingegner’.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegner%')>0;
```

La query seguente limita l’espansione della stringa di testo a due caratteri. Al posto del simbolo di percentuale % utilizzato nella query precedente, viene utilizzato un trattino di sottolineatura (_). Poiché il trattino di sottolineatura è il carattere jolly per il singolo carattere, la stringa di testo non potrà espandersi durante la ricerca oltre un carattere. Ad esempio, la ricerca potrebbe restituire la parola ‘ingegneri’, ma la parola ‘ingegneria’ è troppo lunga per essere restituita.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegner_')>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: le radici delle parole, le coincidenze fuzzy e le coincidenze SOUNDEX.

Ricerca di parole con la stessa radice

In luogo dei caratteri jolly è possibile utilizzare le funzioni di *espansione etimologica* di ConText per espandere l’elenco delle stringhe di testo. Data la “radice” di una parola, ConText espande l’elenco delle parole da cercare sino a includervi tutte le parole che hanno la stessa radice. Di seguito sono riportati esempi di espansioni (questa funzione è pensata in origine per l’uso della lingua inglese).

| <i>Radice</i> | <i>Esempi di espansione</i> |
|---------------|---------------------------------------|
| play | plays played playing playful |
| works | working work worked workman workhorse |
| have | had has haven’t hasn’t |

Dato che ‘works’ e ‘work’ hanno la stessa ‘radice’, una ricerca con espansione etimologica che utilizzi la parola ‘works’ restituirà anche il testo contenente la parola ‘work’.

Per utilizzare l’espansione etimologica all’interno di una query ConText si impiega 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.

Nella query seguente si analizza la tabella PROSPETTIVA alla ricerca di tutti i curriculum che contengono la parola ‘play’ e qualsiasi parola che abbia la stessa radice della parola ‘works’.

```
select Nome
      from PROSPETTIVA
     where CONTAINS(Resume, 'play AND $works')>0;
```

Quando la query precedente viene eseguita, ConText espande la parola ‘works’ sino a comprendere tutte le parole con la stessa radice, quindi esegue la ricerca. Se un curriculum contiene una delle parole con la stessa radice di ‘works’ e contiene anche la parola ‘play’, il record corrispondente verrà restituito all’utente.

L’espansione dei termini di ricerca per mezzo delle radici delle parole semplifica il procedimento di query. Non è più necessario sapere in quale forma un verbo sia stato utilizzato al momento dell’immissione del testo: tutte le forme verbali verranno utilizzate per la ricerca. Non è necessario specificare stringhe di testo particolari, come avviene per le query relative a coincidenze esatte o quando si utilizzano i caratteri jolly. È invece sufficiente specificare una parola, e ConText determinerà dinamicamente tutte le parole che devono essere cercate, basandosi su quella specificata.

Ricerca di coincidenze fuzzy

Una coincidenza fuzzy espande la ricerca specificata sino a includere parole scritte in modo simile, ma che non necessariamente hanno la stessa radice. Le coincidenze fuzzy 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.

Se ad esempio si immette la query seguente, il curriculum di Wilfred Lowell non verrà visualizzato.

```
select Nome
      from PROSPETTIVA
     where CONTAINS(Resume, 'sociale')>0;
```

Il curriculum di Wilfred Lowell non contiene la parola ‘sociale’; Esso però contiene la parola ‘socializza’. Una coincidenza fuzzy restituirà i curriculum contenenti la parola ‘socializza’ anche se ha una radice differente da quella della parola utilizzata come termine di ricerca.

Per effettuare una ricerca fuzzy, occorre far precedere il termine di ricerca da un punto interrogativo (?). Tra il punto interrogativo stesso e l’inizio del termine di ricerca non devono esservi spazi vuoti.

Nell'esempio seguente, la query viene modificata in modo da utilizzare una ricerca fuzzy e restituisce le parole simili alla parola ‘sociale’.

```
select Nome  
      from PROSPETTIVA  
     where CONTAINS(Resume, '?sociale')>0;
```

L'espansione del termine di ricerca utilizzata dalla tecnica della coincidenza fuzzy dipende dal contenuto dell'indice testuale per il testo esaminato. Più voci sono presenti nell'indice testuale e maggiore il numero potenziale di voci restituite dalla coincidenza fuzzy. Per ulteriori informazioni sulla creazione e l'ottimizzazione degli indici testuali si rimanda al Capitolo 30, nel paragrafo “Impostazione dell'opzione ConText”.

Ricerca di parole dal suono simile ad altre parole

Le ricerche a espansione etimologica ampliano la ricerca di un termine fino a comprendere più termini basati sulla ‘radice’ della parola. Le coincidenze fuzzy 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 secondo la pronuncia in inglese. Il metodo di espansione di SOUNDEX utilizza la stessa logica di coincidenza messa a disposizione dalla funzione SQL SOUNDEX.

Per utilizzare l'opzione SOUNDEX all'interno di ConText occorre far precedere il termine di ricerca da un punto esclamativo (!). Tra il punto esclamativo e il termine di ricerca non devono rimanere spazi vuoti. Durante la ricerca, ConText valuta i valori SOUNDEX dei termini presenti nell'indice testuale e cerca tutte le parole che abbiano lo stesso valore.

Nella query seguente vengono cercati tutti i curriculum che contengono la parola inglese ‘guarding’, utilizzando una tecnica di ricerca SOUNDEX.

```
select Nome  
      from PROSPETTIVA  
     where CONTAINS(Resume, '!guarding')>0;
```

Il curriculum di Wilfred Lowell non contiene la parola ‘guarding’, né altre parole con la stessa radice. Quando la query viene eseguita, però, il curriculum viene restituito.

```
set long 1000  
  
select Resume  
      from PROSPETTIVA  
     where CONTAINS(Resume, '!guarding')>0;
```

RESUME

Esperienza con zappa, forcone e vari attrezzi da giardino.
Ottime referenze dal precedente datore di lavoro signor Garden.
Non esagera con le forbici. Socializza con i compagni.
Tra i titoli di studio, una laurea in ingegneria.

Il curriculum di Wilfred Lowell viene restituito dalla ricerca SOUNDEX perché le parole ‘guarding’ e ‘garden’ hanno suono simile in inglese. Utilizzando SQLPLUS è possibile verificare che ambedue le parole hanno lo stesso valore SOUNDEX.

```
select SOUNDEX('guarding') Guarding,
       SOUNDEX('garden') Garden
  from DUAL;
```

```
GUAR GARD
-----
G635 G635
```

Il risultato della query mostra che ambedue le parole ‘guarding’ e ‘garden’ hanno valore SOUNDEX pari a ‘G635.’ Pertanto, una ricerca SOUNDEX di ‘guarding’ con ConText restituirà tutti i record che contengano la parola ‘garden’.

Dato che in inglese ‘garden’ e ‘guarding’ hanno suono simile, ma non hanno lo stesso significato, è necessaria una certa cautela nell’implementazione delle ricerche SOUNDEX. Le ricerche SOUNDEX vengono normalmente utilizzate per trovare nomi di persona, in modo da consentire l’individuazione di nomi simili o che contengano piccole variazioni nell’ortografia. Ad esempio, è possibile utilizzare una ricerca SOUNDEX per individuare curriculum che contengano nomi il cui suono è simile a ‘MacDonald’.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, '!MacDonald')>0;
```

I termini restituiti dalla query comprenderanno i curriculum contenenti il nome MacDonald, insieme a quelli contenenti nomi simili come McDonald, McDonnell e McDonough, in quanto hanno lo stesso valore SOUNDEX.

Combinazione dei metodi di ricerca

I metodi di ricerca disponibili possono essere combinati e annidati. Ad esempio, è possibile combinare due metodi di ricerca per mezzo di un operatore AND.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, 'ingegneria AND guarding')>0;
```

La query può essere modificata in modo da utilizzare un’espansione etimologica sulla parola ‘ingegneria’, come nell’esempio seguente. Occorre aggiungere il simbolo ‘\$’ all’inizio del termine di ricerca da espandere.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, '$ingegneria AND guarding')>0;
```

È quindi possibile modificare la ricerca di ‘guarding’ in una ricerca SOUNDEX.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, '$ingegneria AND !guarding')>0;
```

Quando si esegue questa query, il termine ‘garden’ viene restituito come un termine di ricerca SOUNDEX. Si decide allora di eliminare ‘garden’ come termine di ricerca, per mezzo dell’operatore MINUS. Occorre aggiungere l’operatore MINUS alla funzione CONTAINS, come nell’esempio seguente.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, '($ingegneria AND
      (!guarding MINUS garden))>0;
```

Gli operatori possono inoltre essere annidati, consentendo l’esecuzione di espansioni etimologiche sui termini restituiti da una coincidenza fuzzy. Nell’esempio seguente viene eseguita una ricerca fuzzy della parola ‘guarding’.

I termini restituiti dalla coincidenza fuzzy vengono quindi ampliati utilizzando l’espansione etimologica.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, '$?guarding')>0;
```

Le operazioni di ricerca ConText disponibili sono elencate nella Tabella 29.1.

Tabella 29.1 Operatori di ricerca per ConText.

| 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. |
| {} | 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. |
| ? | Esegue una corrispondenza fuzzy prima della ricerca. |
| ! | Esegue una ricerca SOUNDEX. |
| 0 | Specifica l’ordine in cui sono valutati i criteri di ricerca. |

Limitazione dei record di testo restituiti È possibile specificare il numero massimo di record di testo restituiti da una query. Vengono limitati soltanto i risultati restituiti dalla sezione ConText di una query. Se in una query sono presenti condizioni di limitazione non correlate all'opzione ConText, tali condizioni verranno applicate dopo che il numero di documenti di testo è stato ridotto.

Ad esempio, si supponga che la tabella PROSPETTIVA contenga 200 curriculum. Si selezionano i record dalla tabella basandosi sull'espansione etimologica di una coincidenza fuzzy della parola 'guarding', come nella query seguente.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, '$?guarding')>0;
```

Oltre a utilizzare criteri per la ricerca di testo, si desidera limitare il numero di righe restituite basandosi sul nome della persona. La query viene pertanto modificata in modo da contenere tale criterio.

```
select Nome
  from PROSPETTIVA
 where Nome like 'W%'
   and CONTAINS(Resume, '$?guarding')>0;
```

Le due parti della query verranno elaborate separatamente: la sezione di ricerca del testo dai server ConText e la restrizione sul nome dal normale motore ORACLE di elaborazione delle query. I risultati vengono raccolti e solo i record che soddisfano ambedue i criteri verranno restituiti all'utente.

Per limitare il numero di documenti di testo restituiti, occorre specificare il massimo numero di voci di testo da restituire; queste vengono ordinate per punteggio, con i punteggi più alti (i record più interessanti) per primi. Per specificare il massimo numero di voci di testo da restituire, occorre utilizzare il segno di due punti (:), seguito dal numero di voci. Il massimo numero di voci di testo da restituire può essere aggiunto alla soglia specificata per mezzo della funzione CONTAINS, come nell'esempio seguente.

```
select Nome
  from PROSPETTIVA
 where Nome like 'W%'
   and CONTAINS(Resume, '$?guarding')>0 : 10;
```

In questo caso vengono restituite solo le 10 voci di testo con il punteggio più alto. Ciò non significa che la query restituisca 10 record: è infatti possibile che nessuno dei 10 testi con il punteggio più alto appartenga a un record il cui valore Nome inizia con 'W'.

È inoltre possibile inserire il numero massimo di voci di testo restituite all'interno della funzione CONTAINS, come nell'esempio seguente. In tale esempio la query viene semplificata rimuovendo i criteri addizionali della clausola where e il limite al numero di record restituiti viene posizionato all'interno della clausola di ricerca della funzione CONTAINS.

```
select Nome
  from PROSPETTIVA
 where CONTAINS(Resume, '$?guarding :10')>0;
```

29.3 **Impostazione dell'opzione ConText**

Tutti gli esempi del presente capitolo assumevano che esistesse un indice testuale sulla colonna PROSPETTIVA.Resume e che i server ConText fossero impostati per il database. Nel capitolo seguente è descritta l'impostazione delle tabelle e dei database per supportare l'opzione ConText.

•

•

•

•

• Capitolo 30

•

• **Impostazione dell'opzione ConText**

•

•

-
- 30.1 **Impostazione del database per le ricerche di testo**
-
- 30.2 **Impostazione della tabella per query ConText**
-
- 30.3 **Ottimizzazione degli indici testuali**
- 30.4 **Query in due passaggi**
- 30.5 **Utilizzo dei servizi linguistici**

Per eseguire le ricerche di testo descritte nel capitolo precedente è necessario prima impostare il database e le tabelle per l'utilizzo di ConText. Nel presente capitolo è descritta l'impostazione dell'ambiente di database e degli indici testuali. È inoltre descritta la modifica dell'impostazione di indici testuali e preferenze ConText esistenti. Al termine del capitolo vengono descritti argomenti avanzati, come le query in due passaggi e le opzioni relative ai servizi linguistici.

L'impostazione di ConText all'interno del database richiede l'esecuzione di alcune operazioni da parte dell'amministratore del database (DBA) e dello sviluppatore dell'applicazione. Le operazioni spettanti al DBA devono essere eseguite una sola volta per ciascun database (anche se in seguito è possibile impartire ulteriori comandi DBA per modificare l'impostazione). I comandi spettanti allo sviluppatore dell'applicazione devono essere impartiti ogni volta che una nuova tabella viene abilitata alle ricerche di testo.

30.1 **Impostazione del database per le ricerche di testo**

A livello del database, il DBA deve creare server ConText per l'elaborazione delle sezioni delle query concernenti le ricerche di testo. Il DBA dovrebbe lavorare insieme al team che sviluppa applicazioni basate su ConText per determinare il numero e le categorie (Query, DML, DDL e Linguistic) dei server necessari. Le categorie di comandi supportate da un server forniscono a questo la sua personalità. I server devono essere avviati ogniqualvolta viene avviato il database.

I server ConText possono essere avviati per mezzo di istruzioni della riga di comando (per avviare i server in modalità batch seguendo avvii programmati del database) o interattivamente. Per avviare interattivamente i server ConText è possibile utilizzare l'utilità CTXCTL.

Nell'esempio seguente viene eseguita l'utilità CTXCTL e viene utilizzato il comando help per visualizzare i comandi disponibili.

```
>ctxctl
```

```
*** ConText Option Servers Control ***
```

Servers on machine1.

Type help for a list of commands.

command> **help**

The following commands are available:

| | |
|--|-------------------------|
| help [command] | - commands information |
| status | - show running servers |
| start n [ling query ddl dml]... | - start n servers |
| stop pid... all | - stop server processes |
| quit | - terminate ctxctl |
| exit | - terminate ctxctl |

NOTA Si ringrazia David Grube per la consulenza circa l'architettura ConText.

Per essere in grado di supportare ricerche di testo, è necessario disporre di almeno un server ConText "Query" abilitato all'interno del database. In generale è necessario creare almeno un server ConText per ciascuna delle categorie di server.

I comandi riportati di seguito, se eseguiti all'interno di CTXCTL, avviano i server ConText in modo da supportare tutte le possibili categorie di comandi. Il primo server supporta più categorie di comandi, il secondo supporta solo le query. Quando i comandi vengono eseguiti, viene richiesta l'immissione della password per l'account CTXSYS nel database corrente. Dato che CTXCTL crea dei server nel database corrente, sarebbe bene impostare le variabili di ambiente in modo che puntino a quest'ultimo prima di entrare in CTXCTL.

```
start 1 ddl dml query
start 1 query
```

Per avviare i server ConText dalla riga di comando si utilizza l'utilità CTXSRV. Nell'esempio seguente un server viene avviato con una personalità che lo abilita a supportare quattro categorie di comandi: Q per le query, D per DDL, M per DML e L per i servizi linguistici. Quando si esegue il comando seguente, è necessario sostituire *ctxsys_pass* con la password per l'account CTXSYS.

```
ctxsrv -user ctxsys/ctxsys_pass -personalità QDML
```

Lo stato dei server ConText può essere visualizzato mediante la vista CTS_ALL_SERVERS, come nell'esempio seguente:

```
column Ser_Name format A32
```

```
select Ser_Name,
       Ser_Status,
       Set_Started_At
  from CTX_ALL_SERVERS;
```

Di seguito è riportato un esempio di risultato tratto da CTX_ALL_SERVERS.

| SER_NAME | SER_STAT | SER_START |
|-------------|----------|-----------|
| DRSRV_42736 | IDLE | 03-JUN-97 |

L'esempio mostra che un singolo server ConText è stato avviato nell'istanza. Il nome del server assegnato dal sistema è DRSRV_42736. Per spegnere tutti i server in una sola volta è possibile utilizzare la procedura SHUTDOWN all'interno del package CTX_ADM, come nell'esempio seguente.

```
execute CTX_ADM.SHUTDOWN;
```

È inoltre possibile utilizzare il comando stop all all'interno di CTXCTL per arrestare tutti i server attualmente avviati.

Per arrestare un singolo server ConText occorre specificarne il nome come parametro per l'esecuzione della procedura CTX_ADM.SHUTDOWN, come nell'esempio seguente. Il nome del server è riportato nella colonna Ser_Name di CTX_ALL_SERVERS.

```
execute CTX_ADM.SHUTDOWN('DRSRV_42736');
```

Per visualizzare lo stato dei server ConText sull'host corrente è possibile utilizzare il comando status all'interno di CTXCTL, come nell'esempio seguente.

```
>ctxctl
*** ConText Option Servers Control ***
Servers on machine2.

Type help for a list of commands.

command> status
```

| PID | LING. | QUERY | DDL | DML |
|-------|-------|-------|-----|-----|
| 3576 | | X | | |
| 10809 | | X | | |
| 10812 | | X | | |

| | | | | | |
|-------|---|---|---|---|--|
| 10811 | | X | | | |
| 10816 | | | X | | |
| 10810 | | X | | | |
| 3577 | | X | | | |
| 3578 | | X | | | |
| 3579 | | X | | | |
| 3586 | X | X | X | X | |
| Total | 1 | 9 | 2 | 1 | |

Il risultato mostra che per il server corrente esistono nove server Query, un server Linguistic, due server DDL e un server DML avviati. La colonna PID visualizza i numeri identificativi per i processi server ConText del sistema operativo. I server Query vengono utilizzati per elaborare le sezioni delle query relative a ricerche di testo.

I server DDL vengono utilizzati per elaborare i comandi DDL associati alla creazione degli oggetti database sottostanti, per gli indici testuali. I server DML supportano gli aggiornamenti degli indici testuali. I server Linguistic consentono l'elaborazione linguistica e la generazione di risultati linguistici come i temi e i gist. L'elaborazione linguistica è descritta nel paragrafo “Utilizzo dei servizi linguistici” del presente capitolo.

NOTA Prima di tentare di avviare un server con personalità linguistica è necessario inserire dei dati nella tabella CTXSYS.CTX_SETTINGS. Se in tale tabella non è presente alcuna riga, è possibile utilizzare un file fornito appositamente da Oracle, ctxset.dmp, collocato nella sottodirectory /ctx/admin della home directory del software Oracle. L'operazione di importazione dovrà inserire 11 righe nella tabella. Una volta popolata la tabella CTX_SETTINGS, sarà possibile avviare server ConText linguistici.

Modifiche al file init.ora

Per consentire di eseguire ricerche ConText in un solo passaggio (ovvero che utilizzino la funzione CONTAINS), è necessario impostare nel file init.ora del proprio database il parametro riportato di seguito.

```
text_enable = TRUE
```

NOTA TEXT_ENABLE non è un parametro valido di init.ora per le versioni precedenti ORACLE7.3.

È inoltre possibile impostare il parametro TEXT_ENABLE per la propria sessione mediante il comando alter session, come nell'esempio seguente. Se si utilizza il comando alter session per abilitare le ricerche di testo, si sarà in grado di utilizzare la funzione CONTAINS nelle restanti query della sessione, o sino alla disattivazione del parametro TEXT_ENABLE.

```
alter session set text_enable=TRUE;
```

NOTA *TEXT_ENABLE non è un parametro valido di sessione per le versioni precedenti ORACLE7.3.*

Se si imposta TEXT_ENABLE come TRUE nel file init.ora del database, la ricerca di testo viene abilitata per tutte le sessioni nel database.

Ruoli richiesti

All'interno del database, il dizionario di dati di ConText è di proprietà dell'utente CTXSYS. L'utente CTXSYS è l'unico che dispone del ruolo CTXADMIN. Altri ruoli correlati a ConText comprendono CTXAPP (proprietario dell'applicazione) e CTXUSER (utente dell'applicazione). Il ruolo CTXAPP deve essere concesso a tutti gli utenti che svilupperanno applicazioni ConText. Per consentire all'utente Talbot la creazione di indici testuali su una delle proprie tabelle è necessario concedergli il ruolo CTXAPP, come nell'esempio seguente.

```
grant CTXAPP to Talbot;
```

30.2 Impostazione della tabella per query ConText

La tabella esaminata per mezzo di query di testo deve possedere un vincolo di chiave primaria. ConText utilizza il valore della chiave primaria della tabella per correggere i record restituiti dalla ricerca di testo ai record restituiti da qualsiasi altra condizione di limitazione nella query. Nella versione corrente di ConText la *chiave di testo*, ovvero la chiave primaria della tabella di testo, deve essere una colonna singola. Per la tabella PROSPETTIVA di Talbot non è definita una chiave primaria, anche se la colonna Nome contiene solo valori unici. Negli esempi riportati nel presente capitolo, la colonna Nome viene definita come chiave principale per la tabella PROSPETTIVA.

ConText supporta tre metodi per la ricerca di testo. Negli esempi del capitolo precedente si memorizzavano valori di testo direttamente in una colonna LONG della tabella PROSPETTIVA. In tali esempi il testo viene memorizzato internamente. Un secondo metodo di memorizzazione del testo comporta la memorizzazione *interna master-dettaglio*. La tabella di testo dispone di una tabella di dettaglio e il testo viene memorizzato in quest'ultima.

Le tabelle interne master-dettaglio sono normalmente utilizzate quando i testi sono memorizzati in colonne VARCHAR2. Dato che queste colonne hanno lun-

ghezza limitata, l'utilizzo del metodo di memorizzazione master-dettaglio consente la rappresentazione di un singolo valore di testo per mezzo di più voci VARCHAR2. Il terzo metodo di memorizzazione del testo è quello esterno, in cui il testo viene memorizzato in file esterni al database. Nelle applicazioni a memorizzazione esterna, la tabella di testo contiene le locazioni e i nomi dei file esterni.

La tabella PROSPECTTIVA di Talbot è stata modificata in modo da contenere una colonna LONG di nome Resume. La colonna Resume è stata riempita con puro testo. La tabella contiene ora tre colonne.

| | |
|-----------|--|
| Nome | Nome del candidato all'impiego; VARCHAR2(25). |
| Indirizzo | Indirizzo del candidato all'impiego; VARCHAR2(35). |
| Resume | Curriculum del candidato all'impiego; colonna di tipo LONG, utilizzata per memorizzare puro testo ASCII. |

Per effettuare ricerche di testo sulla colonna Resume della tabella PROSPECTTIVA, è necessario definire una *policy* per la colonna di testo della tabella, ovvero una serie di preferenze che specifica a ConText come deve essere letto e interpretato il testo nella colonna. Ad esempio, può trattarsi di preferenze che consentono ricerche SOUNDEX e specificano la collocazione dei dati.

È possibile creare una policy per una colonna di testo utilizzando le impostazioni predefinite. Sarà quindi possibile modificare la policy stessa dopo la sua creazione. Per creare una policy si esegue la procedura CREATE_POLICY del package CTX_DDL. Per eseguire tale procedura è necessario disporre del ruolo CTXAPP abilitato per la sessione. Per assicurarsi che tutti i ruoli di cui si dispone siano abilitati, è possibile utilizzare il comando set role all.

Nell'esempio seguente viene creata una policy per la colonna Resume della tabella PROSPECTTIVA. La procedura CREATE_POLICY richiede due parametri: il nome della policy e il nome della colonna di testo da indicizzare. Per l'esempio seguente alla policy viene assegnato il nome 'PROSPECTTIVA' e la colonna PROSPECTTIVA.Resume viene specificata come colonna di testo.

```
execute CTX_DDL.CREATE_POLICY('Prospettiva','Prospettiva.resume');
```

Sfortunatamente, per la tabella PROSPECTTIVA il precedente comando fallisce, in quanto su tale tabella non è stato ancora definito un vincolo di chiave primaria. Di seguito è riportato il messaggio di errore visualizzato.

```
begin CTX_DDL.CREATE_POLICY('Prospettiva','Prospettiva.resume'); end;
```

```
*
```

```
ERROR at line 1:
ORA-20000: ConText error:
DRG-10503: suitable textkey not found
ORA-06512: at "CTXSYS.DRUE", line 110
ORA-06512: at "CTXSYS.CTX_DDL", line 833
ORA-06512: at line 1
```

Il messaggio di errore “suitable textkey not found” indica che ConText non è stato in grado di trovare un vincolo di chiave primaria definito per la tabella. Per creare la policy per la tabella PROSPETTIVA, Talbot dovrà prima creare una chiave primaria.

```
alter table Prospettiva add
constraint Prospettiva_pk primary key (Nome);
```

Table altered.

```
execute CTX_DDL.CREATE_POLICY('Prospettiva','Prospettiva.resume');
```

PL/SQL procedure successfully completed.

Così si crea un vincolo di chiave primaria per la colonna Nome della tabella PROSPETTIVA. In seguito viene creata la policy ‘PROSPETTIVA’ per la colonna PROSPETTIVA.Resume.

Una volta creata, la policy ‘PROSPETTIVA’ utilizza le preferenze predefinite per la ricerca ConText. Nel paragrafo seguente sono descritte la visualizzazione e la modifica di tali preferenze.

Visualizzazione e modifica delle preferenze

Per visualizzare le preferenze correnti che costituiscono una policy è possibile eseguire una query sulla vista del dizionario di dati CTX_PREFERENCE_USAGE. Nella query riportata di seguito vengono selezionati da CTX_PREFERENCE_USAGE i nomi delle preferenze (la colonna Pre_Name) per la policy ‘PROSPETTIVA’.

```
select Pre_Name
  from CTX_PREFERENCE_USAGE
 where Pol_Name = 'PROSPETTIVA';

PRE_NAME
-----
DEFAULT_DIRECT_DATASTORE
DEFAULT_NULL_FILTER
DEFAULT_LEXER
NO_SOUND_EX
DEFAULT_STOPLIST
DEFAULT_INDEX
```

Il risultato visualizza le preferenze create per la policy ‘PROSPETTIVA’. Per default la policy assume che i dati siano memorizzati internamente in una singola tabella (la preferenza DEFAULT_DIRECT_DATASTORE), e utilizza stoplist, filtri e opzioni di indicizzazione predefiniti. Se i dati sono memorizzati esternamente in formato binario (come potrebbe avvenire ad esempio per file di MS Word memorizzati in colonne LONG RAW), è possibile utilizzare il filtro Blaster per interpretare il testo nei documenti memorizzati.

Per ulteriori informazioni sui filtri disponibili si rimanda alla *Oracle ConText Option Administrator's Guide*. In ConText 1.1 è disponibile una funzione denominata AUTORECOGNIZE all'interno delle preferenze di filtro (che consente il riconoscimento dei formati dei file memorizzati), la quale non è però supportata nella presente versione.

Una delle preferenze abilitate per la policy ‘PROSPETTIVA’ di Talbot è NO_SOUNDDEX. Per default le ricerche SOUNDDEX non sono abilitate. Per modificare le preferenze della policy è possibile utilizzare la procedura UPDATE_POLICY del package CTX_DLL.

NOTA *Non è possibile aggiornare una policy utilizzata da indici testuali esistenti. Se è già stato creato un indice testuale utilizzando la policy, prima di aggiornare la policy stessa occorre utilizzare la procedura CTX_DDL.DROP_INDEX per scaricare l'indice.*

Al momento di creare la policy ‘PROSPETTIVA’ non sono state specificate preferenze, ma solo il nome della policy e della colonna di testo. Quando si utilizza il package CTX_DDL per specificare preferenze all'interno di policy è necessario adottare una sintassi unica per la definizione dei valori. All'interno della chiamata alla procedura UPDATE_POLICY si utilizza il simbolo ‘=>’ per assegnare un nuovo valore a una preferenza. Ad esempio, per abilitare le ricerche SOUNDDEX l'impostazione della preferenza all'interno dell'esecuzione di UPDATE_POLICY sarà quella riportata di seguito.

```
wordlist_pref => 'CTXSYS.SOUNDDEX'
```

La precedente impostazione di preferenza sarebbe interpretata come “imposta la preferenza wordlist per l'utilizzo della preferenza SOUNDDEX di proprietà di CTXSYS”. CTXSYS è il proprietario del dizionario di dati ConText e possiede preferenze a cui è possibile fare riferimento all'interno delle proprie applicazioni.

Nel seguente blocco PL/SQL viene eseguita la procedura UPDATE_POLICY del package CTX_DDL. Vengono specificati due valori in input per la procedura: il nome della policy (in modo che il database sappia quale policy aggiornare) e il nome della nuova preferenza wordlist.

```
begin
  CTX_DDL.UPDATE_POLICY(policy_name => 'PROSPETTIVA',
                        wordlist_pref => 'CTXSYS.SOUNDDEX');
end;
/
```

PL/SQL procedure successfully completed.

Per verificare la modifica apportata alle preferenze della policy ‘PROSPETTIVA’ basta effettuare una query sulla vista del dizionario di dati CTX_PREFERENCE_USAGE, come nell'esempio seguente.

```
select Pre_Name
  from CTX_PREFERENCE_USAGE
 where Pol_Name = 'PROSPETTIVA';
```

```
PRE_NAME
```

```
DEFAULT_DIRECT_DATASTORE
DEFAULT_NULL_FILTER
DEFAULT_LEXER
SOUNDEX
DEFAULT_STOPLIST
DEFAULT_INDEX
```

Il risultato mostra che la preferenza NO_SOUNDEX è stata sostituita dalla preferenza SOUNDEX. Qualsiasi indice testuale creato utilizzando questa policy ora genererà informazioni di ricerca SOUNDEX e supporterà le ricerche SOUNDEX.

Per scaricare una policy occorre eseguire la procedura DROP_POLICY del package CTX_DLL. Nell'esempio seguente è riportato il comando utilizzabile da Talbot per eliminare la policy 'PROSPETTIVA'.

```
execute CTX_DDL.DROP_POLICY('Prospettiva');
```

Creazione di indici testuali

Un *indice testuale* è una raccolta di tabelle e indici creati e conservati dal database per supportare le ricerche di testo. Prima di creare un indice testuale per una colonna è necessario creare la policy per la colonna stessa, come descritto nei paragrafi precedenti del presente capitolo. La policy determina quali tabelle verranno create come parte dell'indice testuale, i nomi delle tabelle e come queste ultime verranno popolate. Per questo motivo è molto importante che la policy rispecchi correttamente le esigenze relative all'indice testuale, prima che quest'ultimo venga creato.

Per creare un indice testuale è necessario connettersi a ORACLE come proprietario della tabella di testo e disporre del ruolo CTXAPP abilitato per la sessione. Per assicurarsi che tutti i ruoli di cui si dispone siano abilitati è possibile utilizzare il comando set role all.

La procedura CREATE_INDEX del package CTX_DLL impartisce i comandi che creano l'indice testuale. Quando si esegue questa procedura è necessario fornire solo il nome della policy per cui l'indice testuale verrà creato. Quando si è creata la policy sono stati specificati la tabella e il nome della colonna di testo, per cui non è necessario specificare di nuovo tali nomi.

Nell'esempio seguente viene creato un indice testuale utilizzando la policy 'PROSPETTIVA'.

```
execute CTX_DDL.CREATE_INDEX('Prospettiva');
```

```
PL/SQL procedure successfully completed.
```

NOTA *Non è possibile creare un indice testuale se non si crea in precedenza una policy per la colonna di testo.*

La creazione di indici testuali causa la creazione di tabelle e indici all'interno del database. Per questo motivo, per creare un indice testuale è necessario disporre di

almeno un server ConText DDL abilitato per il database stesso. Se non è disponibile alcun server ConText DDL, viene visualizzato il messaggio d'errore riportato di seguito.

```
execute CTX_DDL.CREATE_INDEX('Prospettiva');
begin ctx_ddl.create_index('Prospettiva'); end;

*
ERROR at line 1:
ORA-20000: ConText error:
DRG-10309: insufficient DDL servers running for parallel 1 operation
ORA-06512: at "CTXSYS.DRUE", line 110
ORA-06512: at "CTXSYS.CTX_DDL", line 245
ORA-06512: at line 1
```

L'errore “insufficient DDL servers running” indica che non sono disponibili server ConText DDL per l'elaborazione dei comandi di creazione dell'indice testuale. Se si verifica questo errore, è possibile utilizzare CTXCTL o CTXSRV per avviare un server DDL per il database. Per ulteriori informazioni sulla creazione di server ConText si rimanda al paragrafo “Impostazione del database per le ricerche di testo”, in precedenza nel presente capitolo.

Una volta creato l'indice testuale, è possibile effettuare ricerche di testo nella colonna PROSPETTIVA.Resume, come descritto nel Capitolo 29.

Oggetti di database creati quando si crea un indice testuale, ConText crea tabelle per il supporto delle ricerche di testo specificate dalla policy. Per visualizzare le tabelle create, occorre effettuare una query sulle viste del dizionario di dati.

I nomi assegnati agli oggetti dell'indice testuale saranno basati su un numero unico d'identificazione assegnato alla policy. Per determinare il numero della policy occorre eseguire una query sulla vista del dizionario di dati CTX_USER_POLICIES, come nell'esempio seguente, dove si ottiene il valore Pol_ID della policy ‘PROSPETTIVA’.

```
select Pol_ID,
       Pol_Name
  from CTX_USER_POLICIES;
```

| POL_ID | POL_NAME |
|--------|-------------|
| 1370 | PROSPETTIVA |

```
-----
```

Utilizzando il valore Pol_ID è possibile effettuare una query sulla vista dizionario dati USER_TABLES per individuare le tabelle il cui nome contiene il Pol_ID stesso.

```
select Table_Name
      from USER_TABLES
     where Table_Name like '%1370%';
```

TABLE_NAME

DR_01370_I1T1
DR_01370_I1W
DR_01370_KTB
DR_01370_LST

ConText ha creato quattro tabelle per supportare la policy di ricerca di testo ‘PROSPETTIVA’. La prima tabella, DR_01370_I1T1, memorizza le parole indicizzate e i riferimenti ai documenti. La seconda tabella, DR_01370_I1W, è una tabella opzionale creata solo se nella policy è specificata la preferenza SOUNDEX; detta tabella memorizza parole e relativi valori SOUNDEX. La terza tabella, DR_01370_KTB, mappa chiavi di testo per i documenti indicizzati sugli ID dei documenti. L’ultima tabella, DR_01370_LST, memorizza gli ID dei documenti in elaborazione, modificati o eliminati, insieme all’ID del documento più vicino. I nomi delle tabelle degli indici testuali comprendono sempre i valori di ID della policy.

Quando si crea un indice testuale, ConText crea sempre almeno tre tabelle. La quarta tabella viene creata solamente se si specifica SOUNDEX come preferenza della policy. ConText crea inoltre indici sulle tabelle degli indici testuali. Nell’esempio seguente viene effettuata una query sulla vista del dizionario di dati USER_INDEXES, allo scopo di individuare gli indici creati per le tabelle che supportano la policy ‘PROSPETTIVA’.

```
break on Table_Name

select Table_Name,
       Index_Name
  from USER_INDEXES
 where Table_Name like '%1370%'
 order by Table_Name, Index_Name;
```

TABLE_NAME

INDEX_NAME

| | |
|---------------|---|
| DR_01370_I1T1 | DR_01370_I1I1 |
| DR_01370_I1W | DR_1370_GRP_1_I1I DR_1370_GRP_2_I1I DR_1370_GRP_3_I1I SYS_C0013090 |
| DR_01370_KTB | DR_01370_KID DR_01370_KIK |
| DR_01370_LST | DR_01370_LIX |

8 rows selected.

Degli otto indici creati per supportare l’indice testuale, quattro vengono utilizzati per le colonne della tabella SOUNDEX. Se non si utilizzano ricerche SOUNDEX, l’indice testuale conterrà almeno tre tabelle e quattro indici.

Gli oggetti indici testuali vengono memorizzati nel tablespace predefinito per l’utente, utilizzando i parametri di memorizzazione predefiniti per tale tablespace. Gli indici e le tabelle creati possono essere spostati in tablespace differenti.

È inoltre possibile creare preferenze personalizzate che specifichino gli attributi di memorizzazione per le tabelle degli indici testuali. In generale, è più semplice utilizzare le impostazioni predefinite e spostare gli oggetti una volta che questi sono stati creati. Per spostare gli oggetti indici testuali è necessario essere prima in grado di disattivare i server ConText durante lo spostamento degli oggetti stessi. Per arrestare i server ConText durante la manutenzione degli oggetti è possibile utilizzare l'utilità CTXCTL descritta in precedenza nel presente capitolo. Se non si arrestano i server ConText durante la manutenzione degli oggetti, il database può tentare operazioni di manutenzione dell'indice testuale o accessi durante lo spostamento dell'indice testuale stesso.

Scaricamento di un indice testuale Una policy utilizzata da un indice testuale esistente non può essere aggiornata. Per questo motivo, per modificare una policy è necessario in primo luogo eliminare gli indici testuali che vi fanno riferimento. Per determinare se una policy è stata indicizzata, basta effettuare una query sulla colonna Pol_Status della vista del dizionario di dati CTX_USER_POLICIES.

```
select Pol_Status
  from CTX_USER_POLICIES
 where Pol_Name = 'PROSPETTIVA';
```

Se esiste un indice testuale che utilizza la policy, il valore della colonna Pol_Status sarà 'INDEXED'. Se nessun indice testuale utilizza la policy, il valore della colonna Pol_Status sarà 'NO_INDEX'. Se nessun indice testuale utilizza la policy, sarà possibile scaricare la policy stessa. In tutti i casi, se per la policy esiste un indice testuale, quest'ultimo deve essere scaricato prima della policy.

Se l'indice testuale esiste, è possibile determinare la tabella e la colonna per cui è stato creato. I nomi della tabella e della colonna sono specificati al livello della policy. Pertanto, è possibile effettuare una query su CTX_USER_POLICIES per determinare quali colonne sono comprese nell'indice testuale. La colonna Pol_Key_Name su cui si effettuava una query nell'esempio precedente è la chiave primaria per la tabella. La colonna Pol_Text_Expr visualizza il nome della colonna che contiene il testo.

```
select Pol_TableName,
       Pol_Key_Name,
       Pol_Text_Expr
  from CTX_USER_POLICIES
 where Pol_Name = 'PROSPETTIVA';
```

Di seguito è riportato il risultato per la policy 'PROSPETTIVA'.

POL_TABLENAME

POL_KEY_NAME

POL_TEXT_EXPR

PROSPETTIVA

NAME

RESUME

Per scaricare un indice di testo si utilizza la procedura DROP_INDEX del package CTX_DLL. Per l'esecuzione di questa procedura è necessario fornire come valore in input il nome della policy, in questo caso 'PROSPETTIVA'.

```
execute CTX_DDL.DROP_INDEX('Prospettiva');
```

PL/SQL procedure successfully completed.

Per controllare che l'indice testuale sia stato scaricato occorre verificare se le tabelle dell'indice testuale esistono ancora o meno, effettuando una query sullo stato della policy in CTX_USER_POLICIES, come nell'esempio seguente.

```
select Pol_Status
  from CTX_USER_POLICIES
 where Pol_Name = 'PROSPETTIVA';

POL_STATUS
-----
NO_INDEX
```

Poiché lo stato della policy ora è 'NO_INDEX', non esistono indici testuali che utilizzino correntemente la policy 'PROSPETTIVA'. Quest'ultima può quindi essere scaricata, o modificata per poi creare nuovi indici testuali utilizzando la policy corretta.

30.3 Ottimizzazione degli indici testuali

È possibile ricostruire tutti gli indici testuali per mezzo della procedura OPTIMIZE_INDEX del package CTX_DLL. Questa procedura comprime gli indici e le tabelle frammentati in estensioni più ridotte e rimuove i riferimenti e le informazioni a questi correlate, riguardanti voci di testo modificate o eliminate. Quando una voce di testo viene modificata o eliminata, ConText non modifica o elimina dinamicamente le voci corrispondenti nell'indice testuale; i record sono invece contrassegnati come modificati e l'indice testuale viene aggiornato durante la successiva esecuzione della procedura OPTIMIZE_INDEX.

Per ottimizzare un indice testuale occorre quindi eseguire la procedura OPTIMIZE_INDEX. Questa utilizza come valore in input il nome della policy, come nell'esempio seguente, dove viene ottimizzato l'indice associato alla policy 'PROSPETTIVA'.

```
execute CTX_DDL.OPTIMIZE_INDEX('Prospettiva');
```

Caricamento dei dati

Come si è spiegato in precedenza nel presente capitolo, esistono tre differenti tipi di memorizzazione del testo disponibili in ConText: interna, interna master-dettaglio ed esterna. Per immettere dati nel database è possibile utilizzare SQLPLUS (come

ha fatto Talbot per caricare il curriculum di Wilfred Lowell nella tabella PROSPETTIVA). È inoltre possibile utilizzare SQL*Loader per caricare dati nelle colonne di testo. ConText mette a disposizione un terzo metodo per il caricamento dei dati: l'utilità CTXLOAD. Questa richiede l'immissione del nome utente e della password, il nome della tabella in cui caricare i dati e il nome del file contenente i dati. Il formato dei file CTXLOAD è unico; per ulteriori informazioni su CTXLOAD si rimanda alla *Oracle ConText Option Administrator's Guide*.

30.4 Query in due passaggi

Quando elabora una query di testo, ORACLE procede in due passaggi. Basandosi sui contenuti dell'indice testuale vengono determinate le chiavi di testo corrispondenti dei documenti e i valori delle chiavi testo vengono quindi utilizzati per determinare quale riga soddisfa la ricerca. Nella Figura 30.1 è illustrata la normale elaborazione delle query di testo.

Quando si utilizza la funzione CONTAINS in un'operazione di select, si utilizza una *query a un solo passaggio*. In una query di questo tipo la gestione dell'elenco di chiavi per le corrispondenze di testo viene effettuata dal database. Al momento di eseguire una ricerca di testo, le chiavi per le righe coincidenti vengono memorizzate in una tabella. Una volta completata la query, la tabella viene sottoposta a truncate. Dal punto di vista dell'utente non è necessaria alcuna operazione di pulizia della tabella che conteneva le chiavi di testo.

Le *query in due passaggi* possono essere utilizzate per la manutenzione dell'elenco di chiavi di testo anche una volta completata la query. Una query in due passaggi segue lo stesso percorso generale di esecuzione di una query a un solo passaggio, ma richiede un più alto grado di coinvolgimento dell'utente nell'utilizzo e nella manutenzione delle chiavi di testo risultanti. Nel seguito è descritta l'esecuzione di una query in due passaggi.

Prima di eseguire una query in due passaggi è necessario creare una tabella che conterrà i valori delle chiavi di testo per i documenti che soddisfano i criteri di ricerca. È possibile utilizzare il comando create table dell'esempio seguente; l'unica parte di tale comando che è possibile modificare è il nome della tabella, se si desidera utilizzare un nome differente. Il resto del comando deve essere utilizzato senza alcuna modifica.

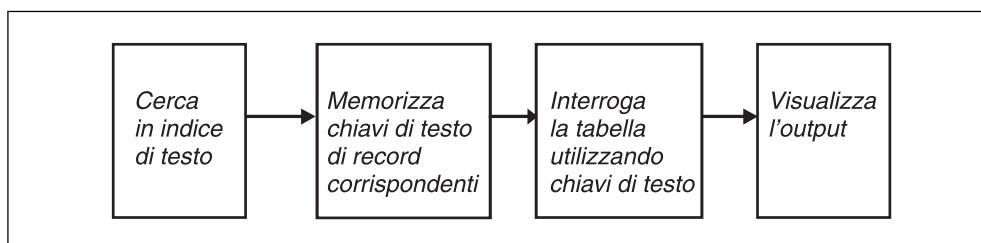


Figura 30.1 Elaborazione delle query di testo.

```
create table CTX_TEMP
(TextKey      VARCHAR2(64),
 Score        NUMBER,
 ConID        NUMBER);
```

Quando si esegue il comando `create table`, nel proprio schema viene creata una tabella denominata `CTX_TEMP`, utilizzata per memorizzare i valori chiave testo delle righe che soddisfano i criteri della ricerca di testo. La colonna `Score` memorizza il punteggio delle ricerche (la misura della fedeltà con cui le ricerche soddisfano i criteri impostati) e la colonna `ConID` agevola l'identificazione dei risultati di una query nel caso che più risultati di query differenti vengano memorizzati nella tabella `CTX_TEMP`.

Per eseguire una query in due passaggi è necessario in primo luogo popolare la tabella `CTX_TEMP` con le chiavi di testo dei documenti che soddisfano i criteri di ricerca. A questo scopo si utilizza la procedura `CONTAINS` del package `CTX_QUERY`. I parametri per la procedura `CONTAINS` sono, nell'ordine, il nome della tabella, i criteri di ricerca e il nome della tabella che conterrà i risultati. Per ulteriori informazioni sui criteri di ricerca si rimanda al Capitolo 29.

Nell'esempio seguente viene eseguita una ricerca di testo e le chiavi testo vengono memorizzate nella tabella `CTX_TEMP`.

```
execute CTX_QUERY.CONTAINS('PROSPETTIVA','ingegneria','CTX_TEMP');
```

Il comando precedente richiama la procedura `CONTAINS` del package `CTX_QUERY`. Viene utilizzata la policy ‘`PROSPETTIVA`’: in questo modo Context utilizza le informazioni della policy per determinare su quale tabella e colonna effettuare la query, in questo caso la colonna `Resume` della tabella `PROSPETTIVA`. Il termine da cercare è ‘`ingegneria`’, anche se in questa clausola sarebbe possibile utilizzare qualsiasi termine di ricerca od operatore. Il risultato della ricerca verrà inserito nella tabella `CTX_TEMP`. I valori delle chiavi testo possono essere selezionati da `CTX_TEMP`.

Riempire la tabella `CTX_TEMP` costituisce il primo passaggio. Il passaggio successivo richiede una query su `CTX_TEMP` insieme alla tabella di testo base per la query. Unendo `CTX_TEMP` alla tabella base (in questo esempio `PROSPETTIVA`), è possibile visualizzare gli attributi delle chiavi di testo ricavati dalla tabella `PROSPETTIVA`. Ad esempio, non è possibile visualizzare il testo selezionato da `CTX_TEMP`; è possibile solamente selezionare i valori delle chiavi testo.

Nell'esempio seguente la tabella `CTX_TEMP` viene unita alla tabella `PROSPETTIVA`. Il valore `CTX_TEMP.TextKey` deve essere uguale al valore della colonna `PROSPETTIVA.Nome`, in quanto `Nome` è definita come chiave primaria della tabella `PROSPETTIVA`. Per visualizzare il testo della colonna `Resume` viene utilizzato il comando `set long`. Senza tale operazione verrebbero visualizzati solo 80 caratteri del curriculum.

```
set long 1000

select PROSPETTIVA.Name,
       PROSPETTIVA.Resume
  from PROSPETTIVA, CTX_TEMP
 where PROSPETTIVA.Name = CTX_TEMP.Textkey
 order by CTX_TEMP.Score desc;
```

Dato che solo uno dei record presenta un valore Resume, e tale curriculum contiene la parola ‘ingegneria’, solo un record viene restituito. Poiché la colonna Resume è lunga, il testo va a capo durante la visualizzazione. All’interno della colonna Resume, i ritorni a capo immessi durante l’inserimento dei dati sono presenti anche nel risultato in output.

NAME

RESUME

WILFRED LOWELL

Esperienza con zappa, forcione e vari attrezzi da giardino.
Ottime referenze dal precedente datore di lavoro signor Garden.
Non esagera con le forbici. Socializza con i compagni.
Tra i titoli di studio, una laurea in ingegneria.

Nella query i record erano ordinati per punteggio.

order by CTX_TEMP.Score desc

Se la query avesse restituito più record, quelli con il punteggio più alto sarebbero stati visualizzati per primi. Maggiore il punteggio e più inherente è il testo selezionato al termine utilizzato per la ricerca.

Dopo l’esecuzione della query è possibile eliminare mediante `delete` i record dalla tabella CTX_TEMP. Non sarà possibile eseguire un’ulteriore query in due passaggi utilizzando la tabella CTX_TEMP sino a quando non si applicherà `commit` all’eliminazione dei record. È possibile utilizzare il comando `truncate` per eliminare tutti i record dalla tabella CTX_TEMP e ridurre la tabella stessa a un’estensione singola.

30.5 Utilizzo dei servizi linguistici

Per supportare query linguistiche su tabelle di testo, ovvero query che comportano temi e “gist” (riassunti) di testo, è necessario che nel database sia avviato un server ConText Linguistic.

Per ulteriori informazioni sulla creazione di server Linguistic da utilizzarsi da parte di ConText si rimanda al paragrafo “Impostazione del database per le ricerche di testo” riportato in precedenza nel presente capitolo. I temi consentono la ricerca per concetti, invece che per parole. In questo modo è possibile cercare testi contenenti il tema ‘banking’ e ottenere la restituzione di documenti relativi a mutui ipotecari, anche se tali documenti non contengono la parola ‘banking’ vera e propria. Un gist mette a disposizione un riassunto intelligente del contenuto testuale. È possibile creare più livelli di gist.

Per elaborare query linguistiche è necessario creare due tabelle per l’elaborazione dei risultati (simili alla tabella richiesta per le query in due passaggi). È possibile utilizzare per le tabelle nomi differenti da quelli riportati nell’esempio, ma è necessario servirsi dell’esatta struttura delle tabelle descritta di seguito.

```

create table CTX_THEMES
(CID      NUMBER,
 PK       VARCHAR2(64),
 Theme    VARCHAR2(256),
 Weight   NUMBER);

create table CTX_GISTS
(CID      NUMBER,
 PK       VARCHAR2(64),
 Pov     VARCHAR2(256),
 Gist    LONG);

```

ConText utilizza la colonna CID per memorizzare i valori ID delle policy (i valori Pol_ID di CTX_USER_POLICIES). Inoltre, ConText utilizza la colonna PK di CTX_THEMES e CTX_GISTS per memorizzare le chiavi testo per i documenti.

Per generare risultati linguistici è necessario eseguire le procedure REQUEST_THEMES e REQUEST_GIST del package CTX_LING.

NOTA *Le procedure REQUEST_THEMES e REQUEST_GIST devono essere avviate una volta per ciascun documento o riga. Per la tabella PROSPETTIVA sarà necessario avviare le procedure una volta per ciascuna riga di dati.*

Nell'esempio seguente viene eseguita la procedura REQUEST_THEMES e vengono utilizzati tre parametri: il nome della policy ('PROSPETTIVA'), il valore della chiave di testo per la riga da analizzare ('WILFRED LOWELL') e la tabella in cui i dati devono essere scritti ('CTX_THEMES'). Viene quindi eseguita la procedura REQUEST_GIST, utilizzando la stessa policy e gli stessi parametri di chiave di testo, ma memorizzando i dati in CTX_GISTS.

```

begin
  CTX_LING.REQUEST_THEMES('PROSPETTIVA','WILFRED LOWELL','CTX_THEMES');
  CTX_LING.REQUEST_GIST('PROSPETTIVA','WILFRED LOWELL','CTX_GISTS');
end;
/

```

PL/SQL procedure successfully completed.

Una volta richiesta l'analisi linguistica, è necessario sottoporre le richieste. Queste verranno gestite da qualsiasi server ConText Linguistic disponibile. Per sottoporre le richieste occorre eseguire la funzione SUBMIT del package CTX_LING. Occorre prima creare una variabile (nell'esempio seguente, 'handle') che conterrà il risultato della funzione.

```

variable handle NUMBER
execute :handle := CTX_LING.SUBMIT

```

PL/SQL procedure successfully completed.

Una volta eseguita la funzione, è possibile visualizzare il valore della variabile 'handle'.

```
print handle
```

```
HANDLE
```

```
-----  
1
```

Ora che i dati linguistici sono stati generati, è possibile unire la tabella CTX_THEMES o CTX_GISTS alla tabella PROSPETTIVA. Ad esempio, è possibile effettuare query su PROSPETTIVA per curriculum che contengono il tema 'work'. Per unire la tabella CTX_THEMES a PROSPETTIVA, occorre unire la colonna CTX_THEMES.PK alla colonna della chiave di testo (Nome) di PROSPETTIVA:

```
where CTX_THEMES.PK = PROSPETTIVA.Nome
```

Nell'esempio seguente è descritta una query che coinvolge CTX_THEMES, PROSPETTIVA e la clausola theme.

```
select Nome,  
       Resume  
  from CTX_THEMES, PROSPETTIVA  
 where CTX_THEMES.PK = PROSPETTIVA.Nome  
   and theme = 'work';
```

I servizi Linguistic di ConText possono essere utilizzati per ampliare la capacità del motore di ricerca di restituire intelligentemente i record corrispondenti. Se possibile, conviene iniziare con query e metodi di ricerca semplici, per poi passare a corrispondenze fuzzy, espansioni sulle radici delle parole e ricerche SOUNDEX a seconda delle necessità. A causa del lavoro aggiuntivo richiesto dalla manutenzione delle tabelle di ricerca e dalla scrittura di query di testo che se ne servano, l'utilizzo dei servizi linguistici dovrebbe essere limitato a quelle applicazioni e a quegli utenti che ne hanno la specifica esigenza. A prescindere dal servizio utilizzato, è necessario controllare costantemente e accuratamente l'utilizzo dei server ConText e dello spazio di memorizzazione degli indici testuali. Se l'applicazione utilizza query in due passaggi o servizi linguistici, è inoltre necessario gestire le esigenze di spazio per la memorizzazione delle chiavi di testo risultanti.

• Capitolo 31

• **Concetti orientati agli oggetti avanzati in ORACLE8**

- 31.1 **Oggetti riga e oggetti colonna**
- 31.2 **Tabelle oggetto e OID**
- 31.3 **Viste oggetto con REF**
- 31.4 **PL/SQL a oggetti**
- 31.5 **Oggetti nel database**

Tutte le funzioni orientate agli oggetti di ORACLE descritte finora nel presente volume condividono due caratteristiche: la qualità di oggetti annidati e la qualità di oggetti colonna. Un *oggetto annidato* è completamente contenuto all'interno di un altro oggetto. Ad esempio, una tabella annidata è contenuta da un'altra tabella ed è pertanto un oggetto annidato. Anche se i dati di una tabella annidata vengono memorizzati separatamente dalla tabella principale, è possibile accedervi solo attraverso la tabella principale stessa. Un *oggetto colonna* viene rappresentato da una colonna di una tabella. Ad esempio, un array variabile è rappresentato da una colonna di una tabella ed è pertanto un oggetto colonna.

Per avvalersi delle caratteristiche orientate agli oggetti un database deve supportare anche *oggetti riga*, oggetti rappresentati come righe invece che come colonne. Gli oggetti riga non sono oggetti annidati, ma *oggetti referenziali*, accessibili per mezzo di riferimenti da altri oggetti. In questo capitolo viene spiegato come creare e referenziare gli oggetti riga.

Come viene spiegato nel presente capitolo, gli oggetti riga possono essere estesi per l'utilizzo con alcuni degli oggetti descritti nei capitoli precedenti (ad esempio le viste oggetto). Oltre agli oggetti riga, è trattata l'applicazione degli oggetti al linguaggio PL/SQL, ovvero la creazione di oggetti PL/SQL.

31.1 Oggetti riga e oggetti colonna

La distinzione tra oggetti riga e oggetti colonna è fondamentale. Si considerino in primo luogo gli oggetti colonna; questi sono basati su caratteristiche già presenti nel database. Un tipo di dati astratto o un array variabile possono essere annidati come una colonna all'interno di una tabella. Una tabella annidata o un LOB possono essere memorizzati in una singola colonna di una tabella. In ciascun 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 utilizza un LOB (Capitolo 27) è necessario specificare i valori di memorizzazione per la tabella che memorizza i dati LOB. ORACLE crea locatori 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 stessa vengono memorizzati. ORACLE crea puntatori dalla tabella principale ai record nelle tabelle annidate. In questo modo, la tabella principale e la sua tabella annidata incorporata vengono correlate.

Negli oggetti riga, gli oggetti vengono rappresentati come righe e non come colonne. I dati non sono incorporati nella tabella principale, ma questa contiene un riferimento a un'altra tabella.

Le tabelle annidate, in quanto memorizzate separatamente dalla tabella principale, mettono a disposizione un utile strumento per la comprensione degli 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 è un oggetto riga? La tabella principale definirà riferimenti ai dati correlati.

Le caratteristiche relazionali tra oggetti di ORACLE descritte nei primi capitoli facevano affidamento sugli oggetti colonna. Nel presente capitolo l'obiettivo è puntato sulle capacità orientate agli oggetti avanzate, basate sugli oggetti riga. Gli oggetti riga vengono utilizzati per creare riferimenti tra righe di differenti tabelle.

31.2 Tabelle oggetto e OID

In una tabella oggetto, ciascuna riga è un oggetto riga. Una tabella oggetto differisce in molti modi da una normale tabella relazionale. In primo luogo, ciascuna riga all'interno della tabella oggetto ha un valore di identificazione oggetto (OID, Object IDentifier) assegnato da ORACLE quando viene creata. Un OID è l'identificatore di un oggetto riga. In secondo luogo, alle righe di una tabella oggetto possono fare riferimento altri oggetti all'interno del database.

Per creare una tabella oggetto si utilizza il comando `create table`. Si consideri il tipo di dati `ANIMALE_TY` descritto nei precedenti capitoli.

```
create or replace type ANIMALE_TY as object
(Famiglia    VARCHAR2(25),
 Nome        VARCHAR2(25),
 DataNascita DATE);
```

NOTA Per mantenere semplice l'esempio, il tipo di dati `ANIMALE_TY` viene creato senza alcuna funzione membro.

Per creare una tabella oggetto del tipo di dati `ANIMALE_TY` si utilizza il comando `create table` riportato di seguito.

```
create table ANIMALE of ANIMALE_TY;
```

Table created.

Questo comando `create table` crea una tabella oggetto. Si noti che il comando ha una sintassi inusuale, in quanto crea la tabella “di” un tipo di dati astratto. La tabella risultante può inizialmente apparire come una normale tabella relazionale.

```
describe ANIMALE
```

| Nome | Null? | Type |
|-------------|-------|--------------|
| FAMIGLIA | | VARCHAR2(25) |
| NOME | | VARCHAR2(25) |
| DATANASCITA | | DATE |

Le colonne della tabella `ANIMALE` sono messe in corrispondenza con gli attributi del tipo di dati `ANIMALE_TY`. Esistono però significative differenze nel modo in cui è possibile utilizzare la tabella, in quanto si tratta di una tabella oggetto. Come si è notato in precedenza nel presente 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 metodo costruttore del suo tipo di dati quando si effettua l’`insert` di dati nella tabella. Poiché la tabella `ANIMALE` è basata sul tipo di dati astratto `ANIMALE_TY`, il metodo costruttore `ANIMALE_TY` può essere utilizzato quando si effettua l’inserimento di valori.

NOTA Per ulteriori informazioni sui metodi costruttori si rimanda ai Capitoli 4 e 25.

Nell’esempio seguente si inseriscono tre righe nella tabella oggetto `ANIMALE`. In ciascun comando viene richiamato il metodo costruttore `ANIMALE_TY`.

```
insert into ANIMALE values
  (ANIMALE_TY('MULO','FRANCES',
              TO_DATE('01-APR-1997','DD-MON-YYYY')));
insert into ANIMALE values
  (ANIMALE_TY('CANE','BENJI',
              TO_DATE('03-SEP-1996','DD-MON-YYYY')));
insert into ANIMALE values
  (ANIMALE_TY('COCCODRILLO','LYLE',
              TO_DATE('14-MAY-1997','DD-MON-YYYY')));
```

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, può essere necessario annidare chiamate a più metodi costruttori all’interno del comando `insert`. Per esempi di tipi di dati astratti anidati si rimanda al Capitolo 4.

NOTA Se una tabella oggetto è basata su un tipo di dati che non contiene tipi di dati annidati, è anche possibile effettuare l'insert di record nella tabella oggetto utilizzando per il comando insert la normale sintassi per le tabelle relazionali.

Quando si effettua l'insert di una riga in una tabella oggetto, ORACLE assegna un ID oggetto (OID) alla riga. La disponibilità di un OID consente di utilizzare la riga come un oggetto a cui è possibile fare riferimento. I valori OID non vengono riutilizzati.

Selezione di valori dalle tabelle oggetto

La tabella ANIMALE, quando è stata creata, si basava interamente sul tipo di dati ANIMALE_TY.

```
create table ANIMALE of ANIMALE_TY;
```

Quando si effettua un select 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. Ciò significa che, se si è utilizzato il tipo di dati ANIMALE_TY come base per una colonna di nome Animale, i nomi degli animali si selezionano con Animale.Nome.

Una tabella oggetto, però, è basata 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 occorre effettuare 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 utilizza un altro tipo di dati astratto per una delle sue colonne, alla colonna di tale tipo di dati si farà riferimento durante tutti i comandi select, update e delete. Per esempi di query da tipi di dati astratti annidati si rimanda al Capitolo 4.

Update e delete da tabelle oggetto

Se la tabella oggetto è basata su un tipo di dati astratto che non utilizza altri tipi di dati astratti, l'update o delete della tabella oggetto utilizza lo stesso formato che si utilizzerebbe per le tabelle relazionali. In questo esempio il tipo di dati ANIMALE_TY non utilizza alcun altro tipo di dati astratto per nessuna delle sue colonne. Per questo motivo, per gli update e delete dalla tabella oggetto ANIMALE ci si comporta come questa fosse una tabella relazionale.

Nell'esempio seguente viene effettuato l'update di una delle righe di ANIMALE.

```
update ANIMALE
  set DataNascita = TO_DATE('01-MAY-1997','DD-MON-YYYY')
 where Nome = 'LYLE';
```

1 row updated.

Si noti che durante l'update si fa riferimento alle colonne della tabella oggetto come se ANIMALE fosse una tabella relazionale. Durante un delete 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 comunque una buona idea.

L'operatore REF

L'operatore REF consente di fare riferimento a oggetti riga esistenti. Nella tabella ANIMALE sono presenti, dopo l'eliminazione di una riga descritta nel paragrafo precedente, due oggetti riga. A ciascun oggetto riga è assegnato un valore OID. Per visualizzare gli OID assegnati è possibile utilizzare l'operatore REF come nell'esempio seguente.

```
select REF(A)
  from ANIMALE A
 where Nome = 'FRANCES';
```

A

000028020915A58C5FAEC1502EE034080009D0DADE15538856
F10606EEE034080009D0DADE100001250000

In questo esempio, alla tabella ANIMALE è stato assegnato l'alias "A" e tale alias è stato utilizzato come dato in input per l'operatore REF. Il risultato visualizza il valore OID dell'oggetto riga che ha soddisfatto le condizioni di limitazione della query.

NOTA L'operatore REF utilizza come dato in input l'alias assegnato alla tabella oggetto. Gli altri operatori che verranno descritti più oltre nel presente capitolo utilizzano anch'essi gli alias come dati in input. Si consiglia pertanto di familiarizzare con la sintassi descritta.

Dato che l'operatore REF può fare riferimento solo a oggetti riga, non è possibile fare riferimento a oggetti colonna. Questi ultimi comprendono tipi di dati astratti, LOB e collezionatori.

Come si può dedurre da quanto esposto finora, l'operatore REF in sé non fornisce alcuna informazione utile. È necessario quindi utilizzare un altro operatore, DEREF, che converte il risultato REF in valori. Sarà quindi possibile utilizzare REF e DEREF per fare riferimento a valori di oggetto riga, come descritto nel paragrafo seguente.

Utilizzo dell'operatore DEREF

L'operatore REF utilizza come argomento un oggetto riga e restituisce un valore di riferimento. L'operatore DEREF ha la funzione opposta: utilizza come argomento un valore di riferimento e restituisce il valore degli oggetti riga. L'operatore DEREF utilizza come argomento il valore OID generato per un riferimento.

In precedenza nel presente capitolo è stata creata la tabella oggetto ANIMALE utilizzando 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 Personae che si prendono cura degli animali. Tale tabella conterrà una colonna per il nome del pastore (NomePastore) e un riferimento alla tabella oggetto ANIMALE (AnimaleCurato), come nell'esempio 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),
```

Nell'ultima riga è però presente una nuova situazione.

```
AnimaleCurato    REF ANIMALE_TY);
```

La colonna AnimaleCurato fa riferimento a dati memorizzati altrove. L'operatore REF punta la colonna AnimaleCurato al tipo di dati ANIMALE_TY. Dato che ANIMALE è la tabella oggetto per il tipo di dati ANIMALE_TY, la colonna AnimaleCurato punta agli oggetti riga all'interno della tabella oggetto ANIMALE. Quando si esegue un describe della tabella PASTORE, è possibile verificare che la sua colonna AnimaleCurato è basata su un REF.

```
describe PASTORE
```

| Name | Null? | Type |
|---------------|-------|--------------|
| NOMEPASTORE | | VARCHAR2(25) |
| ANIMALECURATO | | REF |

Si esegua ora l'insert di un record in PASTORE. Per memorizzare i riferimenti ANIMALE nella colonna AnimaleCurato è necessario utilizzare l'operatore REF.

```
insert into PASTORE
select 'CATHERINE WEILZ',
       REF(A)
  from ANIMALE A
 where Nome = 'BENJI';
```

Si osservi attentamente il comando insert. In primo luogo il valore NomePastore viene selezionato come valore letterale dalla tabella ANIMALE.

```
insert into PASTORE
select 'CATHERINE WEILZ',
```

In seguito deve essere specificato il valore per la colonna AnimaleCurato. Ma il tipo di dati di AnimaleCurato è REF ANIMALE_TY, quindi è necessario selezionare il riferimento dalla tabella oggetto ANIMALE per popolare la colonna AnimaleCurato.

```
REF(A)
from ANIMALE A
where Nome = 'BENJI';
```

Ora, in primo luogo viene eseguita una query sulla tabella oggetto ANIMALE. L'operatore REF restituisce il valore OID per l'oggetto riga selezionato. Questo valore viene memorizzato nella tabella PASTORE come puntatore a tale oggetto riga nella tabella oggetto ANIMALE.

La tabella PASTORE non conterrà le informazioni sull'animale, ma il nome del pastore e un riferimento a un oggetto riga nella tabella ANIMALE. Per visualizzare il valore ODI di riferimento, si esegue una query su PASTORE.

```
select * from PASTORE;
```

| NOMEPASTORE | ANIMALECURATO |
|-----------------|--|
| CATHERINE WEILZ | 000022020815A58C5FAEC2502EE034080009D0DADE15538856 F10606EEE034080009D0DADE |

La colonna AnimaleCurato, come mostra questo esempio, contiene il riferimento all'oggetto riga, non il valore dei dati memorizzati in quest'ultimo.

Il valore a cui si fa riferimento non può essere visualizzato senza utilizzare l'operatore DEREF. Se si effettua una selezione da PASTORE, ORACLE utilizzerà il valore OID per valutare il riferimento (REF). L'operatore DEREF restituirà un valore in base al riferimento.

Nella query seguente il valore della colonna AnimaleCurato in PASTORE è il parametro passato all'operatore DEREF; questo utilizza il valore OID preso da AnimaleCurato per individuare l'oggetto a cui si fa riferimento; l'operatore valuta il riferimento e restituisce 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-96')
```

NOTA Il parametro per l'operatore DEREF è il nome della colonna REF e non il nome della tabella.

Il risultato visualizza che il record ‘CATHERINE WEILZ’ in PASTORE fa riferimento a un record di ANIMALE per l’animale di nome ‘BENJI.’ Alcuni aspetti significativi di questa query meritano di essere sottolineati.

- La query utilizza un riferimento a un oggetto riga per passare da una tabella (PASTORE) a una seconda tabella (la tabella oggetto ANIMALE). Così viene eseguita un’unione in background, senza che vengano specificati criteri di unione.
- La tabella oggetto stessa non viene menzionata nella query. L’unica tabella elenca-ta 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 a cui si fa riferimento e non solo parte della riga.

Tali significative differenze separano le query oggetto dalle normali query relazionali. Così, durante le query sulle tabelle, è necessario sapere come sono Prov sta-bilitate le relazioni tra queste ultime: basandosi su chiavi esterne e chiavi primarie o su tabelle oggetto con tipi di dati REF. Per agevolare il passaggio dall’approccio re-lazionale a quello orientato agli oggetti, ORACLE consente la creazione di viste oggetto che contengono REF sovrapposti alle tabelle relazionali esistenti. Si riman-da al paragrafo “Viste oggetto con REF”, più oltre nel presente capitolo.

L’operatore VALUE

L’operatore DEREF è stato applicato a una tabella relazionale, in questo caso la ta-bella PASTORE. Esso restituisce il valore del riferimento che va dalla tabella rela-zionale alla tabella oggetto.

Che ne è delle query dalla tabella oggetto? È possibile il select da ANIMALE?

| select * from ANIMALE; | | |
|------------------------|---------|-------------|
| FAMIGLIA | NOME | DATANASCITA |
| MULO | FRANCES | 01-APR-97 |
| CANE | BENJI | 03-SEP-96 |

Anche se ANIMALE è una tabella oggetto, è possibile effettuare select da essa come se si trattasse di una tabella relazionale. Ciò è coerente con gli esempi di insert e select descritti in precedenza nel precedente capitolo. Si tratta tuttavia di un'operazione diversa da quella mostrata per mezzo di DEREF. L'operatore DEREF visualizzava la completa struttura del tipo di dati astratto utilizzato dalla tabella oggetto ANIMALE.

```
select DEREF(K.AnimaleCurato)
  from PASTORE K
 where NomePastore = 'CATHERINE WEILZ';

DREF(K.ANIMALECURATO)(FAMIGLIA, NOME, DATANASCITA)
-----
ANIMALE_TY('CANE', 'BENJI', '03-SEP-96')
```

Per visualizzare le stesse strutture mediante una query sulla tabella oggetto ANIMALE occorre utilizzare l'operatore VALUE. Come descritto nell'esempio seguente, l'operatore VALUE visualizza i dati nello stesso formato utilizzato dall'operatore DEREF. Il parametro per l'operatore VALUE è l'alias della tabella.

```
select VALUE(A)
  from ANIMALE A
 where Nome = 'BENJI';

A(FAMIGLIA, NOME, DATANASCITA)
-----
ANIMALE_TY('CANE', 'BENJI', '03-SEP-96')
```

L'operatore VALUE si rivela utile per la risoluzione di problemi di riferimento e all'interno di blocchi PL/SQL, come descritto nel paragrafo "PL/SQL a oggetti", più oltre nel presente capitolo. Dato che viene consentito di effettuare query dei valori formattati direttamente dalla tabella oggetto, tali valori possono essere selezionati senza utilizzare la query DEREF della colonna AnimaleCurato di PASTORE.

Riferimenti non validi

È possibile effettuare il delete dell'oggetto a cui un riferimento punta. Ad esempio, è possibile eliminare una riga dalla tabella oggetto ANIMALE a cui punta un record di PASTORE.

```
delete from ANIMALE
 where Nome = 'BENJI';
```

Il record di PASTORE a cui fa riferimento questo record ANIMALE avrà un cosiddetto *REF erroneamente concordato*. Se si esegue l'insert di una nuova riga ANIMALE per l'animale di nome 'BENJI,' questa non verrà riconosciuta come parte dello stesso riferimento stabilito in precedenza. La ragione sta nel fatto che, la prima volta che si è inserita una riga 'BENJI', ORACLE ha generato un valore OID per l'oggetto riga, e a tale valore faceva riferimento la colonna PASTORE. Quando si elimina con delete l'oggetto riga, il valore OID viene eliminato.

ORACLE non riutilizza i numeri di OID, perciò, quando viene inserito il nuovo record BENJI, a questo viene assegnato un nuovo valore OID, mentre il record di PASTORE punta ancora al vecchio valore.

Esiste una differenza fondamentale tra i sistemi relazionali e i sistemi orientati agli oggetti. In un sistema relazionale l'unione di due tabelle dipende solo dai dati correnti. In un sistema orientato agli oggetti l'unione avviene tra oggetti: solo perché due oggetti hanno gli stessi dati non è possibile stabilire che tali oggetti sono uguali.

31.3 Viste oggetto con REF

Le viste oggetto, come descritto nel Capitolo 25, consentono di sovrapporre strutture orientate agli oggetti a tabelle relazionali esistenti. Ad esempio, è possibile creare tipi di dati astratti e utilizzarli all'interno della vista oggetto di una tabella esistente. L'utilizzo delle viste oggetto consente di accedere alla tabella sia mediante una sintassi per comandi relazionali, sia per mezzo di una 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 25 viene utilizzato nel presente capitolo come parte delle basi necessarie per viste oggetto avanzate. In primo luogo viene creata una tabella CLIENTE, con la colonna Cliente_ID designata come chiave primaria.

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

In seguito 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 nell'esempio 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, sarà necessario utilizzare viste oggetto per accedere ai dati di CLIENTE mediante accessi basati su oggetti (come tramite i metodi). È possibile creare una vista oggetto che specifichi i tipi di dati astratti che si applicano alla tabella CLIENTE. Nell'esempio 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;
```

Nella creazione della vista oggetto CLIENTE_OV vengono specificati i metodi costruttori per i due tipi di dati astratti (INDIRIZZO_TY e PERSONA_TY). È ora possibile accedere alla tabella CLIENTE direttamente (come a una tabella relazionale) o attraverso i metodi costruttori per i tipi di dati astratti.

Nella successiva serie di esempi del presente 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, è possibile utilizzare viste oggetto per creare riferimenti tra le tabelle. ORACLE utilizzerà le relazioni chiave primaria/chiave esterna esistenti per simulare valori OID da utilizzare mediante REF tra le tabelle. Sarà così possibile accedere alle tabelle come a tabelle relazionali o a oggetti. Quando si considerano le tabelle come oggetti, è possibile utilizzare REF per effettuare automaticamente unioni delle tabelle stesse (per alcuni esempi si rimanda al paragrafo “Utilizzo dell'operatore DEREF” riportato in precedenza nel presente capitolo).

La tabella CLIENTE ha una chiave primaria Cliente_ID. Si crei una piccola tabella che conterrà una chiave esterna per il riferimento alla colonna Cliente_ID. Nell'esempio seguente viene creata la tabella CLIENTE_CHIAMATA. La chiave primaria della tabella CLIENTE_CHIAMATA è la combinazione di Cliente_ID e Chiamata_Numer. La colonna Cliente_ID di CLIENTE_CHIAMATA è una chiave esterna nei confronti di CLIENTE: non è possibile effettuare la registrazione di una chiamata per un cliente che non dispone già di un record in CLIENTE. Un singolo attributo, Chiamata_Data, viene creato all'interno della tabella CLIENTE_CHIAMATA.

```
create table CLIENTE_CHIAMATA
(Cliente_ID    NUMBER,
 Chiamata_Numero   NUMBER,
 Chiamata_Data     DATE,
 constraint CLIENTE_CHIAMATA_PK
 primary key (Cliente_ID, Chiamata_Numer),
 constraint CLIENTE_CHIAMATA_FK foreign key (Cliente_ID)
 references CLIENTE(Cliente_ID));
```

Ad esempio, è possibile 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 valori OID ai record di CLIENTE e per generare riferimenti in CLIENTE_CHIAMATA.

Generazione di valori OID Si utilizzi in primo luogo una vista oggetto per assegnare valori OID ai record di CLIENTE. Gli OID vengono assegnati ai record in una tabella oggetto, e una tabella oggetto a sua volta è basata su un tipo di dati astratto. Pertanto è necessario in prima istanza 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);
```

Si crei ora una vista oggetto basata sul tipo CLIENTE_TY, assegnando valori OID ai record in CLIENTE.

```
create view CLIENTE_OV of CLIENTE_TY
with object OID (Cliente_ID) as
select Cliente_ID, Nome, Via, Citta, Prov, Cap
from CLIENTE;
```

View created.

La prima parte di questo comando `create view` assegna alla vista il nome (CLIENTE_OV) e specifica a ORACLE che la struttura della vista stessa è basata sul tipo di dati CLIENTE_TY.

```
create view CLIENTE_OV of CLIENTE_TY
```

La parte successiva del comando `create view` specifica al database come dovranno essere costruiti i valori OID per le righe in CLIENTE. La clausola `with object OID` 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 a cui è possibile fare riferimento, all'interno di una tabella oggetto.

```
with object OID (Cliente_ID) as
```

La parte finale del comando `create view` fornisce la query su cui sarà basato l'accesso ai dati della vista. Le colonne della query devono coincidere con le colonne nel tipo di dati base:

```
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 generati per le righe di CLIENTE_OV sono detti *pkOID* in quanto basati sui valori della chiave primaria.

A qualsiasi tabella relazionale è possibile accedere come a un oggetto riga se si creano viste oggetto opportune.

Generazione dei riferimenti Le righe di CLIENTE_CHIAMATA fanno riferimento a righe di 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. Una volta che l'oggetto CLIENTE_OV è stato creato e che è possibile accedere alle righe in CLIENTE mediante gli OID, è necessario creare valori di riferimento in CLIENTE_CHIAMATA che facciano riferimento a CLIENTE. Posizionati i REF, sarà possibile utilizzare l'operatore DEREF descritto in precedenza nel presente capitolo per accedere ai dati di CLIENTE dall'interno di CLIENTE_CHIAMATA.

Il comando `create view` per la vista oggetto di CLIENTE_CHIAMATA è riportato nell'esempio seguente. In tale comando è presente un nuovo operatore, `MAKE_REF`, che viene descritto dopo l'esempio stesso.

```
create view CLIENTE_CHIAMATA_OV as
select MAKE_REF(CLIENTE_OV, Cliente_ID) Cliente_ID,
       Chiamata_Numer,
       Chiamata_Data
  from CLIENTE_CHIAMATA;
```

Fatta eccezione per l'operazione `MAKE_REF`, tale comando `create view` ha un aspetto del tutto normale. L'operazione `MAKE_REF` è riportata nella riga seguente.

```
select MAKE_REF(CLIENTE_OV, Cliente_ID) Cliente_ID,
```

L'operatore `MAKE_REF` utilizza come argomenti il nome della vista oggetto a 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. Così, i due parametri passati a `MAKE_REF` sono `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 deve essere assegnato un alias di colonna.

L'operatore `MAKE_REF` crea riferimenti (detti `pkREF` in quanto basati su chiavi primarie) dalla vista `CLIENTE_CHIAMATA_OV` alla vista `CLIENTE_OV`. È quindi possibile effettuare 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 ai `REF` della tabella. L'operatore `DEREF` può essere utilizzato per selezionare i valori dei dati cui si fa riferimento, come descritto in precedenza nel presente capitolo. Applicata alle viste oggetto, la query avrà l'aspetto dell'esempio seguente.

```
select DEREF(CCov.Cliente_ID)
  from CLIENTE_CHIAMATA_OV CCov
 where Chiamata_Data = TRUNC(SysDate);

DEREF(CCov.CLIENTE_ID)(CLIENTE_ID, NOME, VIA, CITTA, PROV, CAP)
-----
CLIENTE_TY(123, 'SIGMUND', '47 HAFFNER RD', 'LEWISTON', 'NJ', 22222)
```

Per mezzo della query si è individuato il record in `CLIENTE_CHIAMATA` per cui il valore di `Chiamata_Data` era la data di sistema corrente. Quindi si è preso il valore `Cliente_ID` di tale record e si è valutato il suo riferimento. Il valore `Cliente_ID`, come si è dedotto dall'operatore `MAKE_REF`, puntava a un valore `pkOID` nella vista oggetto `CLIENTE_OV`.

La vista oggetto `CLIENTE_OV` ha restituito il record il cui `pkOID` coincideva con il valore di riferimento. L'operatore `DEREF` ha quindi restituito il valore della riga a cui si faceva riferimento. In questo modo la query ha restituito righe da `CLIENTE` anche se l'utente aveva effettuato una query solo su `CLIENTE_CHIAMATA`.

Le viste oggetto degli oggetti colonna consentono di lavorare con le tabelle come se queste fossero sia tabelle relazionali, sia tabelle relazionali a oggetti. Se estese agli oggetti riga, le viste oggetto consentono di generare valori `OID` basandosi su relazioni prestabilite chiave esterna/chiave primaria. Le viste oggetto consentono di continuare a utilizzare i vincoli esistenti e i comandi `insert`, `update`, `delete` e `select standard`. Inoltre, esse consentono l'utilizzo di caratteristiche orientati agli oggetti come i riferimenti alle stesse tabelle. Le viste oggetto mettono così a disposizione un importante ponte tecnologico per la migrazione verso un'architettura di database di tipo orientato agli oggetti.

Come descritto in precedenza nel presente capitolo, ORACLE esegue unioni che risolvono i riferimenti definiti nel database. Quando si prelevano i dati, questi riportano l'intero oggetto riga a cui si è fatto riferimento.

Per fare riferimento ai dati è necessario stabilire dei `pkOID` nella tabella che rappresenta la chiave primaria nella relazione e utilizzare `MAKE_REF` per generare riferimenti nella tabella che rappresenta la chiave esterna.

Sarà quindi possibile operare sui dati come se questi fossero memorizzati in tabelle oggetto.

31.4 PL/SQL a oggetti

In ORACLE8 i programmi PL/SQL sono in grado di utilizzare i tipi di dati astratti creati dagli utenti. Mentre le precedenti versioni consentivano esclusivamente l'utilizzo dei tipi di dati messi a disposizione da ORACLE (come DATE, NUMBER e VARCHAR2), è ora possibile utilizzare anche tipi di dati personalizzati. Il risultato è una mescolanza di SQL, logica procedurale ed estensioni orientata agli oggetti, detta *PL/SQL a oggetti*.

Il seguente blocco PL/SQL utilizza concetti PL/SQL a oggetti. Il tipo di dati astratto CLIENTE_TY viene utilizzato come tipo di dati per la variabile “Cliente1” e il relativo valore viene popolato per mezzo di una query sulla vista oggetto CLIENTE_OV.

```
set serveroutput on
declare
    Cliente1      CLIENTE_TY;
    ClienteNome   VARCHAR2(25);
    ClienteVia    VARCHAR2(50);
begin
    select VALUE(COV)  into Cliente1
        from CLIENTE_OV COV
        where Cliente_ID = 123;
    ClienteNome := Cliente1.Nome;
    ClienteVia  := Cliente1.Via;
    DBMS_OUTPUT.PUT_LINE(ClienteNome);
    DBMS_OUTPUT.PUT_LINE(ClienteVia);
end;
```

Di seguito è riportato il risultato del blocco PL/SQL.

```
SIGMUND
47 HAFFNER RD
```

PL/SQL procedure successfully completed.

Nella prima parte del blocco PL/SQL vengono dichiarate tre variabili. La prima utilizza il tipo di dati CLIENTE_TY, le altre vengono definite come VARCHAR2. Le variabili “ClienteName” e “ClienteVia” verranno utilizzate come dati in input per la procedura PUT_LINE.

```
declare
    Cliente1      CLIENTE_TY;
    ClienteNome   VARCHAR2(25);
    ClienteVia    VARCHAR2(50);
```

La variabile “Cliente1” viene selezionata dalla vista oggetto CLIENTE_OV (creata nel paragrafo precedente del presente capitolo).

L'operatore VALUE viene utilizzato per prelevare i dati nella struttura del tipo di dati astratto. Poiché i dati verranno selezionati nella struttura del tipo di dati astratto, sarà necessario 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 “ClienteNome” e “ClienteVia” vengono quindi prelevati dagli attributi della variabile “Cliente1” e visualizzati. Non è possibile passare l’intera variabile “Cliente1” alla procedura PUT_LINE.

```

ClienteNome := Cliente1.Nome;
ClienteVia  := Cliente1.Via;
DBMS_OUTPUT.PUT_LINE(ClienteNome);
DBMS_OUTPUT.PUT_LINE(ClienteVia);
end;

```

Questo esempio è volutamente semplice, ma mostra la potenza del PL/SQL a oggetti. Questo linguaggio può essere utilizzato ovunque si utilizzino tipi di dati astratti. Il linguaggio PL/SQL non è più vincolato ai tipi di dati messi a disposizione da ORACLE e può rispecchiare in modo più accurato gli oggetti del database. In questo esempio si è effettuata una query su un oggetto per mostrare come le query siano in grado di accedere sia a oggetti colonna, sia a oggetti riga. È quindi possibile selezionare gli attributi del tipo di dati astratto e manipolarli o visualizzarli. Se si sono definiti metodi per il tipo di dati astratto, è possibile applicare anche questi ultimi.

Ad esempio, è possibile portare i metodi costruttori del tipo di dati all’interno dei blocchi PL/SQL. Nell’esempio seguente viene definita una variabile “NewCliente” utilizzando il tipo di dati CLIENTE_TY. La variabile “NewCliente” viene quindi impostata come uguale a una serie di valori richiamati dal metodo costruttore di CLIENTE_TY. La serie di valori della variabile “NewCliente” viene quindi inserita per mezzo della vista oggetto CLIENTE_OV.

```

declare
    NewCliente  CLIENTE_TY;
begin
    NewCliente := CLIENTE_TY(345,'NewCliente','ViaVal',
    'Citta','ST',00000);
    insert into CLIENTE_OV
    values (NewCliente);
end;

```

Per visualizzare il risultato dell’inserimento, occorre effettuare una query su CLIENTE_OV.

```

select Cliente_ID, Nome from CLIENTE_OV;

CLIENTE_ID  NOME
-----
123  SIGMUND
234  EVELYN
345  NewCliente

```

Oltre a richiamare metodi costruttori, è possibile richiamare i metodi creati sui propri tipi di dati astratti. Se si prevede il confronto tra 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 ulteriori estensioni del PL/SQL a oggetti: si definiscono i tipi di dati e le funzioni al livello del database, e questi saranno richiamabili dall'interno di qualsiasi programma PL/SQL. Il PL/SQL a oggetti rappresenta un miglioramento significativo rispetto al PL/SQL tradizionale.

31.5 Oggetti nel database

Le funzioni disponibili in ORACLE, oggetti colonna, oggetti riga, ed estensioni a oggetti per PL/SQL, consentono l'implementazione di oggetti nel database senza necessità di sacrificare gli investimenti già profusi in analisi e progettazione. È possibile 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 tale livello, sarà possibile 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 è possibile beneficiare delle caratteristiche relazionali di ORACLE. La possibilità di utilizzare all'interno di un'applicazione sia la tecnologia relazionale, sia la tecnologia orientata agli oggetti, consente l'utilizzo dello strumento adeguato per ciascuna operazione all'interno del database.

Per l'implementazione della sezione a oggetti del database relazionale a oggetti si inizia iniziare con la definizione dei tipi di dati astratti che costituiscono il nucleo della propria attività. Ogni caratteristica relazionale a oggetti, sia riferita a oggetti colonna o a oggetti riga, è basata su un tipo di dati astratto. Meglio sono definiti i tipi di dati e i relativi metodi e meglio si sarà in grado di implementare oggetti all'interno del database. Se necessario, è possibile annidare oggetti in modo da disporre di più varianti dello stesso tipo di dati "nocciole". Il risultato sarà un database correttamente progettato in modo da beneficiare sia delle caratteristiche relazionali, sia delle caratteristiche orientate agli oggetti messe a disposizione da ORACLE nella presente versione e nelle versioni successive.

Parte terza

Il dizionario di dati di ORACLE8

• Capitolo 32

• **Guida al dizionario di dati di ORACLE8**

- 32.1 **Nota sulla nomenclatura**
- 32.2 **Le “carte stradali”: DICTIONARY (DICT)
e DICT_COLUMNS**
- 32.3 **Oggetti da cui è possibile selezionare:
tabelle (e colonne), viste, sinonimi e
sequenze**
- 32.4 **Vincoli e commenti**
- 32.5 **Indici e cluster**
- 32.6 **Oggetti specifici di ORACLE8**
- 32.7 **Link di database e snapshot**
- 32.8 **Trigger, procedure, funzioni e package**
- 32.9 **Allocazione e utilizzo dello spazio,
compreso il partizionamento**
- 32.10 **Utenti e privilegi**
- 32.11 **Ruoli**
- 32.12 **Revisioni**
- 32.13 **Monitoraggio: le tabelle V\$ o tabelle
di prestazioni dinamiche**
- 32.14 **Varie**

I dizionario di dati di ORACLE memorizza tutte le informazioni utilizzate per la gestione degli oggetti nel database. Anche se costituisce abitualmente il dominio degli amministratori del database (DBA), il dizionario di dati è una fonte di preziose informazioni anche per gli sviluppatori e gli utenti finali.

Nel presente capitolo, il dizionario di dati viene descritto dal punto di vista dell’utente finale. Per questo motivo le tabelle e le viste del dizionario di dati non sono elencate in ordine alfabetico, ma raggruppate per funzione. Questo tipo di organizzazione consente di individuare rapidamente le informazioni desiderate. Sono descritte le viste del dizionario di dati più utilizzate, con esempi del loro utilizzo.

A seconda delle opzioni di configurazione di ORACLE utilizzate, alcuni dei gruppi non riguardano tutti i database. Le viste più comunemente utilizzate sono elencate per prime. I gruppi utilizzati nel presente capitolo sono elencati di seguito.

1. Le “carte stradali”: DICTIONARY (DICT) e DICT_COLUMNS.
2. Gli oggetti da cui è possibile effettuare selezioni: tabelle (e colonne), viste, sinonimi e sequenze.
3. Vincoli e commenti.
4. Indici e cluster.
5. Oggetti specifici di ORACLE8, come i tipi di dati astratti e i LOB.
6. Link di database e snapshot.
7. Trigger, procedure, funzioni e package.
8. Allocazione e utilizzo dello spazio, compreso il partizionamento.
9. Utenti e privilegi.
10. Ruoli.
11. Auditing.
12. Monitoraggio: le tabelle V\$ o tabelle di prestazioni dinamiche.
13. Varie (CHAINED_ROWS, PLAN_TABLE, interdipendenze, viste riservate al DBA, viste ORACLE Trusted, viste SQL*Loader Direct Load Option, e viste NLS [National Language Support]).

32.1 Nota sulla nomenclatura

Con poche eccezioni, i nomi degli oggetti nel dizionario di dati di ORACLE iniziano con uno dei tre prefissi “USER”, “ALL” o “DBA.” I record nelle viste “USER” contengono di solito informazioni su oggetti di proprietà dell’account che effettua la query. I record nelle viste “ALL” comprendono i record “USER”, più 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. Per la maggior parte degli oggetti del database sono disponibili tutte le viste “USER”, “ALL” e “DBA”.

Mantenendo l’inclinazione della presente guida a privilegiare l’utente, sono particolarmente evidenziate le viste “USER” e quelle accessibili a tutti gli utenti. Le viste “ALL” e “DBA” sono descritte nei casi in cui risultano applicabili alla categoria.

32.2 Le “carte stradali”: DICTIONARY (DICT) e DICT_COLUMNS

Alle descrizioni degli oggetti che costituiscono il dizionario di dati di ORACLE è possibile accedere tramite una vista di nome DICTIONARY. Tale vista, accessibile

anche attraverso il sinonimo pubblico DICT, esegue una query sul database per determinare quale vista del dizionario di dati è possibile visualizzare per l'utente. Vengono inoltre cercati i sinonimi pubblici per tali viste.

Nell'esempio seguente si effettua una query su DICT per ottenere i nomi di tutte le viste del dizionario di dati il cui nome comprende la stringa 'VIEWS'.

La selezione da DICT, come nell'esempio seguente, restituisce il nome dell'oggetto e commenti per ciascun oggetto del dizionario di dati che soddisfi i criteri di ricerca. Nella vista sono presenti solo due colonne: Table_Name e i commenti associati alla tabella (Comments).

```
column Comments format a50

select Table_Name, Comments
  from DICT
 where Table_Name like '%VIEWS%';
```

| TABLE_NAME | COMMENTS |
|------------|---|
| ALL_VIEWS | Description of views accessible to the user |
| DBA_VIEWS | Description of all views in the database |
| USER_VIEWS | Description of the user's own views |

È possibile effettuare query sulle colonne delle viste del dizionario di dati tramite la vista DICT_COLUMNS. Come la vista DICTIONARY, DICT_COLUMNS visualizza i commenti immessi nel database per le viste del dizionario di dati. DICT_COLUMNS comprende tre colonne: Table_Name, Column_Name e Comments. Effettuando una query su DICT_COLUMNS è possibile determinare quali viste del dizionario di dati siano più utili per le proprie attuali esigenze.

Se ad esempio si desidera visualizzare informazioni sull'allocazione e l'utilizzo dello spazio per gli oggetti del database, ma non si sa con sicurezza quali viste del dizionario di dati contengono tali informazioni, è possibile effettuare una query su DICT_COLUMNS come nell'esempio seguente. La query descritta cerca tutte le tabelle del dizionario che contengono una colonna di nome Blocchi.

```
select Table_Name
  from DICT_COLUMNS
 where Column_Name = 'BLOCCHI'
   and Table_Name like 'USER%';
```

| TABLE_NAME |
|---------------------|
| USER_EXTENTS |
| USER_FREE_SPACE |
| USER_SEGMENTS |
| USER_TABLES |
| USER_TAB_PARTITIONS |
| 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 qualvolta non si sa con sicurezza dove trovare i dati desiderati, è sufficiente consultare DICTIONARY e DICT_COLUMNS. Se si trova un grande numero di viste che potrebbero essere utili, occorre effettuare una query su DICTIONARY (come nel primo esempio) per visualizzare i commenti a ciascuna vista.

32.3 Oggetti da cui è possibile selezionare: tabelle (e colonne), viste, sinonimi e sequenze

Alla serie di oggetti di proprietà di un utente ci si riferisce con il termine *catalogo*. Per ciascun utente esiste un solo catalogo. Il catalogo elenca tutti gli oggetti da cui l'utente può effettuare select di record, ovvero gli oggetti che possono essere menzionati nella clausola from di una query. Anche se alle sequenze non si fa riferimento diretto nelle clausole from, ORACLE le comprende nel catalogo. Nel seguito viene spiegato come recuperare informazioni sulle tabelle, le colonne, le viste, i sinonimi, le sequenze e il catalogo utente.

Catalogo: USER_CATALOG (CAT)

Effettuando 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) e la colonna Table_Type ne visualizza il tipo.

```
select Table_Name, Table_Type
  from USER_CATALOG
 where Table_Name like 'T%';
```

| TABLE_NAME | TABLE_TYPE |
|------------|------------|
| TROUBLE | TABLE |
| TWONAME | TABLE |

Alla vista USER_CATALOG è anche possibile fare riferimento per mezzo del sinonimo pubblico CAT.

Sono disponibili due ulteriori cataloghi. ALL_CATALOG elenca tutto quanto contenuto nella vista USER_CATALOG, più qualsiasi oggetto per cui sia stato concesso l'accesso all'utente. In ALL_CATALOG viene inoltre visualizzato qualsiasi oggetto concesso 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 USER_CATALOG, ALL_CATALOG e DBA_CATALOG comprendono una colonna Owner.

Oggetti: USER_OBJECTS (OBJ)

USER_CATALOG visualizza solo informazioni inerenti tipi di oggetti specifici. Per ottenere informazioni su tutti i tipi di oggetti, occorre effettuare una query su USER_OBJECTS. Questa vista può essere utilizzata per ottenere informazioni sui seguenti tipi di oggetti: cluster, link di database, funzioni, indici, package, corpi dei package, procedure, sequenze, sinonimi, tabelle, trigger, viste e nuovi oggetti di ORACLE8 come tipi, corpi dei tipi, directory e librerie. Le colonne USER_OBJECTS sono elencate nella Tabella 32.1.

Tabella 32.1 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 via dicendo). |
| 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. |

USER_OBJECTS (meglio nota con il suo sinonimo pubblico OBJ) contiene numerose informazioni vitali che non sono presenti in altre viste del dizionario di dati. In essa sono registrate le date di creazione (colonna Created) e di ultima modifica degli oggetti (l'ultima colonna Last_DDL_Time). Tali colonne sono molto utili quando si tenta di riconciliare diverse serie di tabelle nella stessa applicazione.

Nell'esempio seguente si seleziona la data di creazione e la data di ultima modifica di più oggetti.

```
column Last_DDL_Time format a13

select Object_Name, Created, Last_DDL_Time
  from USER_OBJECTS
 where Object_Name like 'T%';

OBJECT_NAME      CREATED      LAST_DDL_TIME
-----          -----
TROUBLE          07-AUG-01   14-AUG-01
TWONAME          07-AUG-01   07-AUG-01
```

Questo esempio visualizza la data in cui gli oggetti sono stati creati e la data in cui gli oggetti sono stati modificati per l'ultima volta. I dati nell'esempio mostrano che due oggetti sono stati creati lo stesso giorno, ma la tabella TROUBLE è stata modificata in seguito.

NOTA *Se si ricreano oggetti con qualsiasi metodo, ad esempio tramite Import, i relativi valori Created cambieranno, indicando la data di ultima creazione.*

Sono disponibili due ulteriori elenchi di oggetti. ALL_OBJECTS elenca tutto quanto contenuto nella vista USER_OBJECTS, più qualsiasi oggetto a cui si abbia diritto di accesso. Inoltre, in ALL_OBJECTS vengono visualizzati eventuali oggetti il cui accesso sia stato concesso come PUBLIC. DBA_OBJECTS è una versione a livello di DBA dell'elenco di oggetti, che visualizza tutti gli oggetti del database. Oltre alle colonne Table_Name e Table_Type visualizzate 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 ulteriori informazioni su un oggetto è necessario visualizzare la vista specifica per il tipo dell'oggetto. Per le tabelle, tale vista è USER_TABLES ed è possibile farvi riferimento con il sinonimo pubblico TABS.

NOTA *Nelle precedenti versioni di ORACLE era presente una vista detta TAB. Tale vista, simile per funzionamento a TABS, è ancora supportata a causa dei prodotti ORACLE che vi fanno riferimento. La vista TAB però non contiene le colonne contenute da TABS. Nelle query sul dizionario di dati occorre utilizzare TABS.*

Le colonne in USER_TABLES possono essere suddivise in quattro categorie principali (“Identificazione”, “Correlate allo spazio”, “Correlate alle statistiche” e “Altre”), come illustrato nella Tabella 32.2.

Il nome della tabella compare nella colonna Table_Name. La colonna Backed_Up 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 di dati correlate alle partizioni sono descritte nel paragrafo “Allocazione e utilizzo dello spazio, compreso il partizionamento” del presente capitolo).

Nel caso di una tabella oggetto, la colonna Table_Type_Owner visualizza il proprietario del tipo.

Le colonne “correlate allo spazio” sono descritte nel paragrafo dedicato alla memorizzazione della guida alfabetica di riferimento (Capitolo 37). Le colonne “correlate alle statistiche” sono popolate quando la tabella viene analizzata (si veda il comando analyze nella guida alfabetica di riferimento del Capitolo 37).

Tabella 32.2 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 |
| Table_Type | Pct_Used | Avg_Space | Table_Lock |
| Table_Type_Owner | Ini_Trans | Chain_Cnt | |
| Packed | Max_Trans | Avg_Row_Len | |
| IOT_Name | Initial_Extent | Sample_Size | |
| Logging | Next_Extent | Last_Analyzed | |
| IOT_Type | Min_Exts | Avg_Space_Freelist_Blocks | |
| Temporary | Max_Exts | Num_Freelist_Blocks | |
| Nested | Pct_Increase | | |
| | Freelists | | |
| | Freelist_Groups | | |

Effettuando una query sulla colonna Table_Name di USER_TABLES vengono visualizzati i nomi di tutte le tabelle nell'account corrente. Nell'esempio seguente sono elencate tutte le tabelle i cui nomi iniziano con la lettera 'L'.

```
select Table_Name from USER_TABLES
where Table_Name like 'L%';
```

TABLE_NAME

```
-----
LEDGER
LOCATION
LODGING
LONGTIME
```

La vista ALL_TABLES visualizza tutte le tabelle di proprietà dell'utente e le eventuali tabelle per le quali è stato garantito l'accesso all'utente stesso. 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. Tale vista, essendo in grado di contenere voci per più utenti, contiene, oltre alle colonne elencate nella Tabella 32.2, una colonna Owner. DBA_TABLES, che elenca tutte le tabelle del database, per quanto riguarda le colonne ha la stessa definizione di ALL_TABLES.

Le colonne Degree e Instances nella categoria "Altre" della Tabella 32.2 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 il paragrafo dedicato a create table nella guida alfabetica di riferimento del Capitolo 37.

Colonne: USER_TAB_COLUMNS (COLS)

Anche se gli utenti non effettuano query sulle colonne, la vista del dizionario di dati che visualizza le colonne è strettamente legata alla vista del dizionario di dati delle tabelle. Tale vista, detta USER_TAB_COLUMNS, elenca informazioni specifiche sulle colonne. A USER_TAB_COLUMNS è possibile fare riferimento per mezzo del 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 32.3.

Tabella 32.3 Colonne di USER_TAB_COLUMNS.

| IDENTIFICAZIONE | CORRELATE ALLA DEFINIZIONE | CORRELATE ALLE STATISTICHE |
|--------------------|----------------------------|----------------------------|
| Table_Name | Data_Type | Num_Distinct |
| Column_Name | Data_Length | Low_Value |
| Column_ID | Data_Precision | High_Value |
| Packed | Data_Scale | Density |
| Character_Set_Name | Nullable | Num_Nulls |
| | Default_Length | Num_Buckets |
| | Data_Default | Last_Analyzed |
| | Data_Type_Mod | Sample_Size |
| | Data_Type_Owner | |

Le colonne Table_Name e Column_Name contengono i nomi delle tabelle e colonne. L'utilizzo delle colonne “correlate alla definizione” è descritto nel paragrafo “Tipi di dati” della guida alfabetica di riferimento del Capitolo 37. Le colonne “correlate alle statistiche” sono popolate quando la tabella viene analizzata (si consulti il comando analyze nella guida alfabetica di riferimento del Capitolo 37).

Altre statistiche sulle colonne vengono messe a disposizione nella vista USER_TAB_COL_STATISTICS (descritta brevemente), disponibile in ORACLE8.

Per visualizzare le definizioni delle colonne per una tabella, occorre effettuare una query su USER_TAB_COLUMNS specificando il 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 possono anche essere ottenute per mezzo del comando SQLPLUS describe. Questo però non consente di visualizzare, se desiderato, le impostazioni predefinite e le statistiche della colonna. Se si utilizzano tipi di dati astratti, describe non visualizza i nomi dei tipi di dati utilizzati. USER_TAB_COLUMNS, invece, visualizza i nomi dei tipi di dati.

La vista ALL_TAB_COLUMNS visualizza le colonne di tutte le tabelle e viste di proprietà dell'utente e di tutte le tabelle e viste per cui all'utente è garantito l'accesso. Tale vista, essendo in grado di contenere voci per più utenti, contiene, oltre alle colonne elencate nella Tabella 32.3, una colonna Owner. DBA_TAB_COLUMNS, che elenca le definizioni delle colonne di tutte le tabelle del database, per quanto riguarda le colonne ha la stessa definizione di ALL_TAB_COLUMNS.

Statistiche sulle colonne In ORACLE8 la maggior parte delle statistiche sulle colonne sono state spostate da USER_TAB_COLUMNS a USER_TAB_COL_STATISTICS. Le colonne disponibili in USER_TAB_COL_STATISTICS sono elencate nella Tabella 32.4.

Tabella 32.4 Colonne di USER_TAB_COL_STATISTICS.

| NOME COLONNA | DEFINIZIONE |
|---------------|---|
| Table_Name | Nome della tabella. |
| Column_Name | Nome della colonna. |
| Num_Distinct | Numero di valori distinti nella colonna. Disponibile anche in USER_TAB_COLUMNS per compatibilità con le vecchie versioni. |
| Low_Value | Il valore più basso della colonna. Disponibile anche in USER_TAB_COLUMNS per compatibilità con le vecchie versioni. |
| High_Value | Il valore più alto della colonna. Disponibile anche in USER_TAB_COLUMNS per compatibilità con le vecchie versioni. |
| Density | Densità della colonna. Disponibile anche in USER_TAB_COLUMNS per compatibilità con le vecchie versioni. |
| Num_Nulls | Numero di valori NULL nella colonna. |
| Num_Buckets | Numero di bucket utilizzati per l'istogramma della colonna durante l'analisi. |
| Sample_Size | Dimensioni del campione utilizzato per analizzare la colonna. |
| Last_Analyzed | Data di ultima analisi della colonna. |

USER_TAB_COL_STATISTICS contiene colonne statistiche che sono anche fornite da USER_TAB_COLUMNS, per compatibilità con le vecchie versioni. Anche se a tali colonne è possibile accedere da USER_TAB_COLUMNS, è consigliato accedervi tramite USER_TAB_COL_STATISTICS.

Istogrammi sui valori delle colonne A partire da ORACLE7.3 è possibile utilizzare istogrammi per migliorare l'analisi dell'ottimizzatore basato sui costi. La vista USER_TAB_HISTOGRAMS contiene informazioni sull'istogramma di ciascuna colonna, compresi Table_Name, Column_Name, Bucket_Number ed Endpoint_Value.

I valori di USER_TAB_HISTOGRAMS vengono utilizzati dall'ottimizzatore per determinare la distribuzione dei valori delle colonne all'interno della tabella. Di USER_TAB_HISTOGRAMS sono disponibili versioni "ALL" e "DBA".

Colonne aggiornabili A partire da ORACLE7.3 è possibile aggiornare record delle viste che contengono unioni nelle rispettive query. La vista USER_UPDATABLE_COLUMNS elenca tutte le colonne che è possibile aggiornare. È possibile effettuare una query di Owner, Table_Name, e Column_Name per la colonna. Le colonne Updatable avranno il valore 'YES' se possono essere aggiornate e il valore 'NO' altrimenti. È inoltre possibile effettuare una query per determinare se sia consentito inserire o eliminare record dalla vista tramite le colonne Insertable e Deletable.

Viste: USER_VIEWS

Alla query di base per una vista è possibile accedere tramite la vista del dizionario di dati USER_VIEWS, che contiene nove colonne. Di seguito sono elencate le tre colonne principali.

| | |
|-------------|---|
| View_Name | Nome della vista. |
| Text_Length | Lunghezza in caratteri della query di base della vista. |
| Text | Query utilizzata dalla vista. |

Le ulteriori sei colonne, descritte più oltre, sono correlate alle viste oggetto.

La colonna Text è di tipo LONG. Ciò può costituire un problema quando si effettuano query su USER_VIEWS mediante SQLPLUS, in quanto in SQLPLUS i LONG vengono troncati. Il punto in cui avviene il troncamento può essere impostato dall'utente. USER_VIEWS dispone di un meccanismo per determinare l'adeguata impostazione del punto di troncamento LONG, descritto nell'esempio seguente.

La colonna Text_Length visualizza la lunghezza della query della vista. Pertanto il punto di troncamento dei LONG in SQLPLUS deve essere impostato su un valore maggiore o uguale al valore Text_Length della vista. Ad esempio, nell'esempio seguente è visualizzata una vista il cui View_Name è AGING e la cui Text_Length è pari a 500.

```
select View_Name, Text_Length
  from USER_VIEWS
 where View_Name = 'AGING';
```

| View_Name | Text_Length |
|-----------|-------------|
| AGING | 500 |

Dato che la lunghezza del testo della vista è pari a 500 caratteri, è necessario utilizzare il comando set long riportato nell'esempio seguente per portare il punto di troncamento LONG sino ad almeno 500 (il valore predefinito è 80). In caso contrario non sarà possibile visualizzare il testo completo della query per la vista.

```
set long 500
```

Una volta impostato correttamente il punto di troncamento LONG, sarà possibile effettuare una query su USER_VIEWS per ottenere il testo della vista, nel modo seguente:

```
select Text
  from USER_VIEWS
 where View_Name = 'AGING';
```

Se non si fosse utilizzato il comando `set long`, il risultato in output sarebbe stato troncato a 80 caratteri, senza che alcun messaggio avvertisse del fatto. Prima di effettuare query su altre viste sarà necessario controllare i rispettivi valori di `Text_Length`.

Le definizioni delle colonne per le viste possono essere visualizzate effettuando una query su `USER_TAB_COLUMNS`, la stessa vista su cui è possibile effettuare query per le tabelle.

Se nelle viste si utilizzano alias per le colonne, e questi sono parte della query sulla vista, le query sul dizionario di 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.

È inoltre possibile creare viste utilizzando la sintassi riportato di seguito.

```
create view RIVISTA_VISTA (UnArticolo, UnaSezione)
  as select Articolo, Sezione
    from RIVISTA;
```

Elencando i nomi delle colonne nel titolo del comando `create view` si rimuovono gli alias per le colonne dalle query, evitando così la loro visualizzazione mediante `USER_VIEWS`. L'unico modo per visualizzare i nomi delle colonne della vista sarebbe quello di effettuare una query su `USER_TAB_COLUMNS`. Se i nomi delle colonne si trovano nella query, basta solamente effettuare una query su una vista del dizionario di dati (`USER_VIEWS`), sia per la query, sia per i nomi delle colonne.

Data ad esempio la vista `RIVISTA_VISTA` creata nell'esempio precedente, effettuando una query su `USER_VIEWS` viene visualizzato quanto riportato nell'esempio seguente.

```
select Text
  from USER_VIEWS
 where View_Name = 'RIVISTA_VISTA';
```

```
TEXT
-----
select Articolo, Sezione from RIVISTA
```

Questa query non visualizza i nuovi nomi assegnati alle colonne, in quanto tali nomi non sono stati resi parte della query sulla vista. Per visualizzare detti nomi delle colonne in `USER_VIEWS`, è necessario aggiungerli come alias di colonna alla query della vista.

```
create view RIVISTA_VISTA (UnArticolo, UnaSezione)
  as select Articolo UnArticolo, Sezione UnaSezione
    from RIVISTA;
```

Se 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 UnArticolo, Sezione UnaSezione
  from NEWSPAPER
```

Per supportare le viste oggetto, ORACLE8 mette a disposizione sei nuove colonne, disponibili in USER_VIEWS.

| | |
|------------------|---|
| Type_Text | Clausola tipo della vista tipo. |
| Text_Type_Length | Lunghezza della clausola tipo 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 | Proprietario 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 25 e 31.

La vista ALL_VIEWS elenca tutte le viste di proprietà dell'utente e le viste per le quali all'utente è stato garantito l'accesso. Questa vista, essendo 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, per quanto riguarda le colonne ha la stessa definizione di ALL_VIEWS.

Sinonimi: USER_SYNONYMS (SYN)

La vista USER_SYNONYMS elenca tutti i sinonimi di proprietà dell'utente. Di seguito sono descritte le colonne di tale vista.

| | |
|--------------|--|
| Synonym_Name | Nome del sinonimo. |
| Table_Owner | Proprietario della tabella a cui il sinonimo fa riferimento. |
| Table_Name | Nome della tabella a cui il sinonimo fa riferimento. |
| DB_Link | Nome del link di database utilizzato nel sinonimo. |

USER_SYNONYMS è utile per il debugging dei programmi o la risoluzione di problemi relativi agli accessi degli utenti a oggetti all'interno di applicazioni. A USER_SYNONYMS è possibile fare riferimento con il sinonimo pubblico SYN.

Se il sinonimo non utilizza un link di database, la colonna DB_Link ha valore NULL. Pertanto, per visualizzare un elenco dei link di database correntemente utiliz-

zati dai sinonimi posseduti dal proprio account è possibile eseguire la query riportata nell'esempio seguente.

```
select distinct DB_Link
  from USER_SYNONYMS
 where DB_Link is not null;
```

Lo stesso tipo di query può essere effettuato dai DBA a livello del database, utilizzando invece la vista DBA_SYNONYMS.

La vista ALL_SYNONYMS elenca tutti i sinonimi di proprietà dell'utente e i sinonimi per i quali all'utente è stato garantito l'accesso. Questa vista, essendo in grado di contenere entry per più utenti, contiene, oltre alle colonne elencate in precedenza, una colonna Owner. DBA_SYNONYMS, che elenca tutti i sinonimi del database, per quanto riguarda le colonne ha la stessa definizione di ALL_SYNONYMS.

Sequenze: USER_SEQUENCES (SEQ)

Per visualizzare gli attributi delle sequenze è possibile effettuare una query sulla vista del dizionario di dati USER_SEQUENCES. Su tale vista è possibile inoltre effettuare una query utilizzando il sinonimo pubblico SEQ. Nella Tabella 32.5 sono elencate le colonne della vista USER_SEQUENCES.

Tabella 32.5 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 ALL_SEQUENCES elenca tutte le sequenze di proprietà dell'utente e quelle per le quali all'utente è stato concesso l'accesso. Questa vista, essendo in grado di contenere entry per più utenti, contiene, oltre alle colonne elencate nella Tabella 32.5, una colonna Owner. DBA_SEQUENCES, che elenca tutte le sequenze del database, per quanto riguarda le colonne ha la stessa definizione di ALL_SEQUENCES.

32.4 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. Tipici vincoli sono NOT NULL, UNIQUE, PRIMARY KEY e FOREIGN KEY. Nei paragrafi seguenti viene descritto come recuperare informazioni dal dizionario di dati sui vincoli e i commenti.

Vincoli: USER_CONSTRAINTS

Le informazioni sui vincoli possono essere ottenute per mezzo della vista USER_CONSTRAINTS. Tali informazioni sono molto utili quando si tenta di modificare i vincoli dei dati o di risolvere problemi con i dati di un'applicazione. Le colonne contenute in questa vista sono elencate nella Tabella 32.6.

Tabella 32.6 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) |
| 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; 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. |
| Last_Change | Ultima data in cui il vincolo è stato abilitato o disabilitato. |

Questa vista, pur essendo di tipo “USER”, contiene una colonna Owner. Tale colonna si riferisce in questo caso 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 32.6. La comprensione dei tipi di vincoli è fondamentale quando si tenta di ottenere informazioni utili sui vincoli stessi.

I vincoli FOREIGN KEY hanno sempre valori per le colonne R_Owner e R_Constraint_Name in USER_CONSTRAINTS. Tali due colonne specificano il vincolo a cui FOREIGN KEY fa riferimento. Un FOREIGN KEY fa riferimento a un altro vincolo e non a un’altra colonna.

USER_CONSTRAINTS elenca tutti i vincoli di una tabella. I vincoli NOT NULL sulle colonne sono memorizzati come CHECK e dispongono pertanto di un Constraint_Type uguale a ‘C’.

Effettuando una query su USER_CONSTRAINTS si ottengono i nomi di tutti i vincoli di una tabella. Ciò è importante per l’interpretazione di messaggi di errore che riportano solo il nome del vincolo violato.

Una volta noto il nome e il tipo di un vincolo, è possibile verificare le colonne a esso associate tramite la vista USER_CONS_COLUMNS, descritta nel paragrafo seguente.

Ad esempio, si supponga di voler sapere quali colonne fanno parte della PRIMARY KEY di una determinata tabella. Allo scopo, è prima necessario identificare il nome del vincolo PRIMARY KEY della tabella stessa. La tabella riportata di seguito restituisce l’informazione desiderata.

```
select Owner, Constraint_Name
  from USER_CONSTRAINTS
 where Table_Name = 'COMPLEANNO'
   and Constraint_Type = 'P';
```

| OWNER | CONSTRAINT_NAME |
|--------|-----------------|
| TALBOT | SYS_C0008791 |

Il nome Constraint_Name visualizzato nell’esempio è generato dal sistema. Ai vincoli può anche essere assegnato un nome, nel momento della loro creazione. Per ulteriori informazioni si rimanda al paragrafo “Vincoli di integrità” nella guida alfabetica di riferimento del Capitolo 37.

Se il nome di un vincolo è stato generato dal sistema, ciò viene indicato nella colonna Generated. Se si differisce un vincolo, esso non verrà imposto per la durata di una transazione. Se ad esempio si desidera effettuare aggiornamenti su vasta scala delle tabelle correlate e non è possibile garantire l’ordine delle transazioni, si possono differire i controlli effettuati dai vincoli sulle tabelle sino al completamento di tutti gli aggiornamenti.

ALL_CONSTRAINTS elenca i vincoli su tutte le tabelle a cui l’utente è in grado di accedere. DBA_CONSTRAINTS elenca tutti i vincoli del database. Ciascuna di queste viste ha la stessa serie di colonne di USER_CONSTRAINTS: tutte contengono la colonna Owner.

Colonne vincolo: USER_CONS_COLUMNS

Le colonne associate ai vincoli possono essere visualizzate per mezzo della vista del dizionario di dati USER_CONS_COLUMNS. Se si è già effettuata una query su USER_CONSTRAINTS per ottenere i tipi e i nomi dei vincoli coinvolti, è possibile utilizzare USER_CONS_COLUMNS per determinare quali colonne sono comprese nel vincolo. Le colonne contenute in questa vista sono descritte nella Tabella 32.7.

Tabella 32.7 Colonne di USER_CONS_COLUMNS.

| NOME COLONNA | DESCRIZIONE |
|-----------------|--|
| 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 | Posizione della colonna all'interno della definizione del vincolo. |

In questa vista sono presenti solo due colonne non contenute in USER_CONSTRAINTS: Column_Name e Position. Nell'esempio seguente è riportato un modello di query su tale tabella, utilizzando il Constraint_Name ricavato dall'esempio relativo a USER_CONSTRAINTS.

```
select Column_Name, Position
  from USER_CONS_COLUMNS
 where Owner = 'TALBOT'
   and Constraint_Name = 'SYS_C0008791';
```

| COLUMN_NAME | POSITION |
|-------------|----------|
| NOME | 1 |
| COGNOME | 2 |

Come è desumibile dall'esempio precedente, la combinazione di Nome e Cognome forma la chiave primaria per la tabella COMPLEANNO.

La colonna Position è significativa. In ORACLE, quando si crea un vincolo UNIQUE o PRIMARY KEY, il database crea automaticamente un indice unico sulla serie di colonne di dati. L'indice viene creato basandosi sull'ordine delle colonne specificato. A sua volta l'ordine delle colonne influisce sulle prestazioni dell'indice: un indice che comprenda più colonne non verrà utilizzato dall'ottimizzatore a meno che la colonna principale di tale indice (Position='1') sia utilizzata nella clausola where della query. Per ulteriori informazioni sull'ottimizzatore, si rimanda al Capitolo 36.

Le viste ALL_CONS_COLUMNS e DBA_CONS_COLUMNS hanno la stessa definizione delle colonne di USER_CONS_COLUMNS. ALL_CONS_COLUMNS può essere utilizzata per visualizzare informazioni delle colonne relative ai vincoli su tutte le tabelle a 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. Ad esempio, è possibile tentare di creare un vincolo PRIMARY KEY in una colonna che contiene lo stesso valore per più record. Il tentativo di creare un vincolo PRIMARY KEY su una simile colonna fallirebbe.

ORACLE consente di ottenere informazioni sulle righe che causano il fallimento della creazione di vincoli. In primo luogo è necessario creare una tabella di nome EXCEPTIONS nel proprio schema. Lo script SQL da utilizzarsi per creare tale tabella si chiama UTLEXCPT.SQL e si trova normalmente nella directory /rdbms/admin nella home directory di ORACLE.

Una volta creata la tabella EXCEPTIONS, si può tentare di attivare il vincolo descritto nell'esempio:

```
alter table RIVISTA enable PRIMARY KEY
exceptions into EXCEPTIONS;
```

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 il vincolo violato è stato creato) e Constraint (il vincolo violato dalla riga). È pertanto possibile confrontare il Row_ID nella tabella EXCEPTIONS con il RowID nella tabella in cui il vincolo doveva essere attivato (nell'esempio precedente, la tabella RIVISTA).

Se ad esempio il vincolo PRIMARY KEY dell'esempio precedente genera eccezioni, è possibile effettuare una query sulla tabella EXCEPTIONS come nell'esempio seguente.

```
select Owner, Table_Name, Constraint from EXCEPTIONS;
```

| OWNER | TABLE_NAME | CONSTRAINT |
|--------|------------|------------|
| TALBOT | RIVISTA | SYS_C00516 |
| TALBOT | RIVISTA | SYS_C00516 |

Il risultato della query mostra che due righe violano il vincolo SYS_C00516 (che nell'esempio è il nome assegnato al vincolo PRIMARY KEY per la tabella RIVISTA). È possibile determinare quali righe della tabella RIVISTA corrispondano a tali eccezioni effettuando l'unione della colonna Row_ID della tabella EXCEPTIONS con la pseudocolonna RowID della tabella RIVISTA.

```
select *
  from RIVISTA
 where RowID in
       (select Row_ID from EXCEPTIONS);
```

Vengono così visualizzate le righe che causano la violazione del vincolo.

Commenti di tabella: USER_TAB_COMMENTS

I commenti possono essere aggiunti a tabelle, viste o colonne dopo la loro creazione. I commenti alle viste del dizionario di dati sono la base per i record visualizzati mediante le viste DICTIONARY e DICT_COLUMNS. Per visualizzare i commenti sulle proprie tabelle, occorre utilizzare la vista USER_TAB_COMMENTS.

La vista USER_TAB_COMMENTS contiene tre colonne.

| | |
|------------|---|
| Table_Name | Nome della tabella o vista. |
| Table_Type | Tipo dell'oggetto (tabella, tabella oggetto o vista). |
| Comments | Commenti immessi per l'oggetto. |

Si esegua una query su USER_TAB_COMMENTS specificando il Table_Name per cui si desidera visualizzare i commenti, come nell'esempio seguente.

```
select Comments
      from USER_TAB_COMMENTS
     where Table_Name = 'COMPLEANNO';
```

Per aggiungere un commento a una tabella, si utilizza il comando comment come nell'esempio seguente.

```
comment on table COMPLEANNO is 'Date di compleanno degli impiegati';
```

Per eliminare un commento da una tabella, basta specificarlo come due apici senza spazio tra di essi (''), come nell'esempio seguente.

```
comment on table COMPLEANNO is '';
```

Per visualizzare i commenti di tutte le tabelle a cui è possibile accedere, si utilizza la vista ALL_TAB_COMMENTS. Tale vista contiene una colonna Owner aggiornale che specifica il proprietario della tabella. DBA_TAB_COMMENTS elenca tutte le tabelle nel database e prevede anch'essa una colonna Owner.

Commenti di colonna: USER_COL_COMMENTS

USER_COL_COMMENTS visualizza i commenti immessi per colonne all'interno delle tabelle. Tali commenti vengono aggiunti al database per mezzo del comando comment. USER_COL_COMMENTS contiene tre colonne.

| | |
|-------------|----------------------------------|
| Table_Name | Nome della tabella o vista. |
| Column_Name | Nome della colonna. |
| Comments | Commento immesso per la colonna. |

Si esegua una query su USER_COL_COMMENTS specificando Table_Name e Column_Name per cui si desidera visualizzare i commenti, come nell'esempio seguente.

```
select Comments
      from USER_COL_COMMENTS
```

```
where Table_Name = 'COMPLEANNO'
  and Column_Name = 'ETA';
```

Per aggiungere un commento a una colonna, si utilizza il comando `comment` come nell'esempio seguente.

```
comment on column COMPLEANNO.ETA is 'Età in anni';
```

Per eliminare un commento da una colonna, basta specificarlo come due apici senza spazio tra di essi (''), come nell'esempio seguente.

```
comment on column COMPLEANNO.ETA is '';
```

Per visualizzare i commenti di tutte le colonne a cui è possibile accedere, si utilizza la vista `ALL_COL_COMMENTS`. Tale vista contiene una colonna `Owner` aggiornata che specifica il proprietario della colonna. `DBA_COL_COMMENTS` elenca tutte le colonne in tutte le tabelle nel database e prevede anch'essa una colonna `Owner`.

32.5 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 viene descritto l'utilizzo delle viste del dizionario di dati che descrivono indici e cluster. Nel paragrafo "Allocazione e utilizzo dello spazio, compreso il partizionamento" del presente capitolo sono utilizzate descrizioni di viste del dizionario di 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 avviene per i vincoli, esistono due viste del dizionario di dati utilizzate per ottenere informazioni sugli indici: `USER_INDEXES` e `USER_IND_COLUMNS`. A `USER_INDEXES` è anche possibile fare riferimento mediante il sinonimo pubblico `IND`.

Le colonne contenute in `USER_INDEXES` possono essere suddivise in quattro categorie, elencate nella Tabella 32.8.

Il nome dell'indice viene riportato nella colonna `Index_Name`. Il proprietario e il nome della tabella indicizzata vengono visualizzati nelle colonne `Table_Owner` e `Table_Name`. La colonna `Uniqueness` contiene '`UNIQUE`' per indici unici e '`NONUNIQUE`' per indici non unici. La colonna `Table_Type` specifica se l'indice riguarda una tabella o un cluster.

Le colonne "correlate allo spazio" sono descritte nel paragrafo dedicato alla memorizzazione della guida alfabetica di riferimento del Capitolo 37. Le colonne "correlate alle statistiche" sono popolate quando la tabella viene analizzata (si consulti il comando `analyze` nella guida alfabetica di riferimento del Capitolo 37). Le colonne `Degree` e `Instances` fanno riferimento al grado di parallelismo utilizzato al momento di creare l'indice.

Tabella 32.8 Colonne di USER_INDEXES.

| 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 | |
| Uniqueness | Next_Extent | Avg_Data_Blocks_Per_Key | |
| Status | Min_Exts | Clustering_Factor | |
| Partitioned | Max_Exts | Num_Rows | |
| Index_Type | Pct_Increase | Sample_Size | |
| Temporary | Pct_Free | Last_Analyzed | |
| Generated | Freelists | | |
| Logging | Freelist_Groups | | |
| | Pct_Threshold | | |

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 nell'esempio seguente.

```
select Index_Name, Uniqueness
  from USER_INDEXES
 where Table_Owner = 'TALBOT'
   and Table_Name = 'COMPLEANNO';
```

| INDEX_NAME | UNIQUENESS |
|---------------|------------|
| ----- | ----- |
| PK_COMPLEANNO | UNIQUE |

Il risultato della query nell'esempio precedente mostra che sulla tabella COMPLEANNO esiste un indice di nome "PK_COMPLEANNO". Utilizzando le stesse convenzioni per l'attribuzione dei nomi a tutti gli indici è possibile rendere molto facile l'identificazione dello scopo e della tabella cui si riferisce un indice, semplicemente osservando il valore Index_Name. In questo esempio "PK" significa PRIMARY KEY; il nome del vincolo può essere specificato durante la creazione dello stesso 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 tali righe e più efficienti saranno le *query d'intervallo*, ovvero le query in cui per una colonna viene dato un intervallo di valori, come in where Cognome like 'A%'.

Per sapere quali colonne fanno parte dell'indice e conoscere il loro ordine all'interno dell'indice stesso è necessario effettuare una query sulla vista USER_IND_COLUMNS, descritta nel paragrafo seguente.

ALL_INDEXES visualizza tutti gli indici di proprietà dell'utente e tutti gli indici creati su tabelle a cui l'utente stesso ha accesso. Questa vista, essendo in grado di contenere entry per più utenti, contiene, oltre alle colonne descritte nella Tabella 32.8, una colonna **Index_Owner**. **DBA_INDEXES**, che elenca tutti gli indici del database, per quanto riguarda le colonne ha la stessa definizione di **ALL_INDEXES**.

NOTA Le versioni “*ALL*” e “*DBA*” della vista **USER_INDEXES** prevedono un’ulteriore colonna per la definizione del proprietario dell’indice, ma tale colonna si chiama *Index_Owner*, non *Owner*.

Colonne indicizzate: **USER_IND_COLUMNS**

Per determinare quali colonne sono presenti in un indice occorre effettuare una query su **USER_IND_COLUMNS**. Le colonne disponibili in tale vista sono elencate nella Tabella 32.9.

Tabella 32.9 Colonne di **USER_IND_COLUMNS**.

| NOME COLONNA | DESCRIZIONE |
|-----------------|--|
| Index_Name | Nome dell’indice. |
| Table_Name | 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. |

In questa vista sono presenti solo tre colonne non contenute in **USER_INDEXES**: **Column_Name**, **Column_Position** e **Column_Length**. **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** nella guida alfabetica di riferimento del Capitolo 37). Nell’esempio seguente è riportato un modello di query su questa tabella, che utilizza l’**Index_Name** dell’esempio relativo a **USER_INDEXES**. In questo esempio alla colonna **Column_Position** viene 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 può essere rilevato dalla query, l’indice **PK_COMPLEANNO** consiste in due colonne: nell’ordine, Nome e Cognome.

L'ordine delle colonne determina la possibilità di utilizzare l'indice per determinate query. Se gli utenti utilizzano di frequente la colonna Cognome nelle proprie clausole where e non utilizzano la colonna Nome nelle stesse query, l'indice può necessitare di una riorganizzazione. Un indice che comprenda più colonne non verrà utilizzato dall'ottimizzatore a meno che la colonna principale di tale indice (Column_Position='1') sia utilizzata nella clausola where della query. Per ulteriori informazioni sull'ottimizzatore si rimanda al Capitolo 36.

Le viste ALL_IND_COLUMNS e DBA_IND_COLUMNS per quanto riguarda le colonne hanno la stessa definizione di USER_IND_COLUMNS. ALL_IND_COLUMNS può essere utilizzata per visualizzare informazioni sulle colonne relative agli indici sulle tabelle a cui l'utente è in grado di accedere, a prescindere dal proprietario. La vista DBA_IND_COLUMNS contiene le informazioni sugli indici a livello di colonna per l'intero database.

Cluster: USER_CLUSTERS (CLU)

Ai parametri di memorizzazione e statistiche associati ai cluster è possibile accedere tramite USER_CLUSTERS (a cui è possibile fare riferimento anche mediante il sinonimo pubblico CLU). Le colonne presenti in tale vista del dizionario di dati sono descritte nella Tabella 32.10, suddivise per tipo.

Tabella 32.10 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 | | |
| | Ini_Trans | | |
| | Max_Trans | | |
| | Initial_Extent | | |
| | Next_Extent | | |
| | Min_Exts | | |
| | Max_Exts | | |
| | Pct_Increase | | |
| | Freelists | | |
| | Freelist_Groups | | |

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'utilizzo delle colonne "correlate allo spazio" è descritto nel paragrafo dedicato alla memorizzazione della guida alfabetica di riferimento del Capitolo 37. Le colonne "correlate alle statistiche" sono popolate quando la tabella viene analizzata (si veda il comando analyze nella guida alfabetica di riferimento del Capitolo 37).

Le colonne Degree e Instances raggruppate sotto "Altre" nella Tabella 32.10 sono relative al modo in cui le query che coinvolgono il cluster vengono messe in parallelo all'interno di un'istanza e attraverso le istanze. Per la descrizione delle clausole degree, instances e cache per i cluster si rimanda al paragrafo dedicato a create cluster nella guida alfabetica di riferimento del Capitolo 37.

Non esiste una vista "ALL_CLUSTERS". La vista DBA_CLUSTERS prevede una colonna aggiuntiva Owner in quanto elenca tutti i cluster del database.

A partire da ORACLE7.2, è possibile specificare una funzione di hash per un cluster. Le funzioni di hash specificate dall'utente sono più utili se: (1) la colonna chiave del cluster è numerica; (2) i valori della colonna chiave del cluster vengono assegnati sequenzialmente; (3) si conosce il massimo numero di valori di colonna chiave del cluster per la tabella. Se le tre condizioni sono soddisfatte è possibile utilizzare la clausola hash is del comando create cluster per specificare proprie funzioni di hashi (si consulti il paragrafo dedicato a create cluster nella guida alfabetica di riferimento del Capitolo 37).

Colonne cluster: USER_CLU_COLUMNS

Per visualizzare la corrispondenza delle colonne della tabella con le colonne del cluster occorre effettuare una query su USER_CLU_COLUMNS. Di seguito sono elencate le colonne presenti in quest'ultima.

| | |
|-----------------|--|
| 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 la determinazione di quali colonne, provenienti da quali tabelle, siano mappate sulle 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 la mappatura delle colonne di tutti i cluster del database.

32.6 Oggetti specifici di ORACLE8

In ORACLE8 sono disponibili numerosi tipi di oggetti nuovi. Nei paragrafi seguenti vengono descritte le viste del dizionario di dati associate con ciascuno degli oggetti di nuovo tipo.

Tipi di dati astratti

I tipi di dati astratti creati all'interno di uno schema sono elencati nella vista del dizionario di dati USER_TYPES. Questa comprende colonne per il nome del tipo (Type_Name), il numero di attributi (Attributes) e il numero di metodi (Methods) definiti per il tipo di dati.

Ad esempio, il tipo di dati ANIMALE_TY dispone di tre attributi e un metodo (i metodi vengono definiti per mezzo del comando create type body), come si vede nell'esempio seguente.

```
select Type_Name,
       Attributes,
       Methods
  from USER_TYPES;
```

| TYPE_NAME | ATTRIBUTES | METHODS |
|------------|------------|---------|
| ANIMALE_TY | 3 | 1 |

Per visualizzare gli attributi di un tipo di dati è necessario effettuare una query sulla vista del USER_TYPE_ATTRS. Le colonne presenti nella vista USER_TYPE_ATTRS sono elencate nella Tabella 32.11.

Tabella 32.11 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. |

È possibile effettuare una query su USER_TYPE_ATTRS per visualizzare le relazioni tra tipi di dati astratti annidati. Ad esempio, il tipo di dati PERSONA_TY utilizza il tipo di dati INDIRIZZO_TY, come si vede nell'esempio seguente.

```
select Attr_Name,
       Length,
       Attr_Type_Name
  from USER_TYPE_ATTRS
```

```
where Type_Name = 'PERSON_TY';
```

| ATTR_NAME | LENGTH | ATTR_TYPE_NAME |
|-----------|--------|----------------|
| NOME | 25 | VARCHAR2 |
| INDIRIZZO | | ADDRESS_TY |

Se per un tipo sono definiti dei metodi, sarà possibile effettuare una query su USER_TYPE_METHODS per determinare i nomi dei metodi stessi. USER_TYPE_METHODS contiene 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 di cui è stato eseguito l'overloading) 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.

Ad esempio, il tipo di dati ANIMALE_TY possiede un metodo, una funzione membro di nome ETA. Tale metodo prevede un parametro in input (Datanascita) e un risultato in output (l'età, in giorni). Una query su USER_TYPE_METHODS mostra che per ETA sono definiti due parametri, in luogo dell'unico parametro atteso.

```
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 in input?. Per scoprirllo è possibile effettuare 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 ciascun metodo ORACLE ha creato un parametro隐式的“SELF”. È possibile utilizzare tale parametro durante l'esecuzione del metodo o affidarsi al valore predefinito.

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 |

Se si utilizzano REF, la vista del dizionario di dati USER_REFS mostra quelli definiti. Questa vista mostra il nome della tabella contenente la colonna REF (Table_Name) e il nome di tabella per la colonna oggetto (Column_Name). Anche agli attributi dei REF, che specificano ad esempio se il REF è memorizzato con o senza RowID, è possibile accedere mediante USER_REFS.

I tipi collettori (tabelle annidate e array variabili) sono descritti per mezzo della vista del dizionario di dati USER_COLL_TYPES. Le colonne contenute nella vista USER_COLL_TYPES sono elencate nella Tabella 32.12.

Tabella 32.12 Colonne di USER_COLL_TYPES.

| NOME COLONNA | DESCRIZIONE |
|--------------------|---|
| TYPE_NAME | Nome del tipo. |
| COLL_TYPE | Tipo di collettore. |
| UPPER_BOUND | Per collettori ARRAY VARIABILI, massimo numero di valori. |
| ELEM_TYPE_MOD | Modificatore del tipo per il collettore. |
| ELEM_TYPE_OWNER | Proprietario del tipo utilizzato dal collettore. |
| ELEM_TYPE_NAME | Nome del tipo utilizzato dal collettore. |
| LENGTH | Massima lunghezza di un elemento di tipo di dati carattere. |
| PRECISION | Precisione dell'elemento. |
| SCALE | Scala dell'elemento. |
| CHARACTER_SET_NAME | Nome del set di caratteri dell'elemento. |

La vista USER_COLL_TYPES può essere utilizzata in combinazione con le viste del dizionario di dati per i tipi di dati astratti descritte in precedenza nel presente paragrafo, allo scopo di determinare la struttura del tipo di un collettore.

Di tutte queste viste del dizionario di dati sono disponibili versioni “ALL” e “DBA”. Dato che tali viste contengono record per più proprietari, le versioni “ALL” e “DBA” delle viste stesse prevedono, oltre alle colonne elencate nel presente paragrafo, anche colonne Owner.

LOB

Come descritto nel Capitolo 27, all'interno del database è possibile memorizzare oggetti di grandi dimensioni (LOB, Large Objects). La vista del dizionario di dati USER_LOBS fornisce informazioni sui LOB definiti nelle proprie tabelle, come nell'esempio 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 inoltre i nomi dei segmenti utilizzati per contenere i dati LOB quando questi aumentano. La vista non visualizza però i tipi di dati delle colonne LOB. Per visualizzare i tipi di dati dei LOB è possibile effettuare un `describe` della tabella oppure eseguire una query su `USER_TAB_COLUMNS` come descritto in precedenza nel presente capitolo.

Per utilizzare tipi di dati BFILE per i LOB è necessario creare delle directory (si consulti il paragrafo dedicato a `create directory` nella guida alfabetica di riferimento del Capitolo 37). Il dizionario di dati `ALL_DIRECTORIES` visualizza i valori di Owner, Directory_Name e Directory_Path per ciascuna directory a cui sia consentito l'accesso. È inoltre disponibile una versione “DBA” di questa vista, mentre non è disponibile una versione “USER”.

32.7 Link di database e snapshot

Link di database e snapshot vengono utilizzati per gestire l'accesso ai dati remoti. A seconda del tipo di snapshot utilizzato, può essere anche possibile utilizzare i log di snapshot. Nei paragrafi seguenti sono descritte le viste del dizionario di dati che possono essere utilizzate per visualizzare informazioni su link di database, snapshot e log di snapshot.

Per ulteriori informazioni sui link di database si rimanda al Capitolo 21. Per ulteriori informazioni sugli snapshot si rimanda al Capitolo 28.

Link di database: `USER_DB_LINKS`

Per visualizzare i link di database creati nel proprio account occorre effettuare una query su `USER_DB_LINKS`. Tale vista, le cui colonne sono descritte nella Tabella 32.13, visualizza informazioni sulla connessione remota per stabilire la quale verrà utilizzato il collegamento. I valori Username e Password sono utilizzati per connettersi al database remoto definito dal valore Host.

La colonna Host memorizza i descrittori di connessione SQL*Net. Tale colonna contiene l'esatta stringa di caratteri specificata durante l'esecuzione del comando `create database link` e non modifica le maiuscole e le minuscole. È pertanto necessario prestare attenzione alle maiuscole del testo utilizzato per la creazione di link di database, altrimenti le query su `USER_DB_LINKS` dovranno tenere conto di incoerenze tra le maiuscole e le minuscole nella colonna Host.

Tabella 32.13 Colonne di USER_DB_LINKS.

| NOME COLONNA | DESCRIZIONE |
|--------------|--|
| DB_Link | Nome del link di database. |
| Username | Nome utente da utilizzare nel database remoto. |
| Password | Password per il nome utente nel database remoto. |
| Host | Stringa SQL*Net da utilizzare per la connessione al database remoto. |
| Created | Data per la creazione del link di database. |

Ad esempio, per cercare un link di database che utilizzi il descrittore di servizio ‘HQ’ occorre immettere quanto segue:

```
select * from USER_DB_LINKS
where UPPER(Host) = 'HQ';
```

Infatti è possibile che esistano voci con valore di Host uguale a ‘hq’ invece di ‘HQ’.

NOTA *Se si utilizzano connessioni predefinite al database remoto, Password sarà NULL.*

La vista ALL_DB_LINKS elenca tutti i link di database che sono di proprietà dell’utente o PUBLIC. DBA_DB_LINKS elenca tutti i link nel database. ALL_DB_LINKS e DBA_DB_LINKS condividono la maggior parte delle definizioni per le colonne con USER_DB_LINKS. I primi hanno però una colonna Owner al posto della colonna Password.

Per ulteriori informazioni sugli utilizzi dei link di database, si rimanda al Capitolo 21.

Snapshot: USER_SNAPSHOTS

È possibile effettuare una query su USER_SNAPSHOTS per visualizzare informazioni sugli snapshot di proprietà del proprio account. Tale vista, le cui colonne sono elencate nella Tabella 32.14, visualizza informazioni strutturali sugli snapshot complete dei relativi intervalli di aggiornamento.

Alla creazione di uno snapshot vengono creati numerosi oggetti di database. Nel database locale, ORACLE crea una tabella che viene popolata con i dati della tabella remota e in più, per gli snapshot semplici, i RowID o le chiavi primarie per tali dati. ORACLE crea quindi una vista di tale tabella, utilizzando il nome dello snapshot come nome della vista. Per ulteriori informazioni sugli snapshot si rimanda al Capitolo 28.

Quando si effettua una query sullo snapshot, in realtà si effettua una query sulla vista corrispondente. Il nome della vista è riportato nella colonna Name di USER_SNAPSHOTS.

Tabella 32.14 Colonne di USER_SNAPSHOTS.

| NOME COLONNA | DESCRIZIONE |
|---------------------|---|
| Owner | Account proprietario dello snapshot. |
| Name | Nome dello snapshot. |
| Table_Name | Tabella base (nel database locale) per lo snapshot. |
| Master_View | Nome della vista utilizzata durante gli aggiornamenti dello snapshot. |
| Master_Owner | Account proprietario della tabella base (nel database remoto) per lo snapshot. |
| Master | Tabella base (nel database remoto) per lo snapshot |
| Master_Link | Link di database utilizzato per accedere al database master. |
| Can_Use_Log | Flag che indica se lo snapshot è in grado di utilizzare un log ('YES' o 'NO'). |
| Updatable | Flag che indica se lo snapshot può essere aggiornato. |
| Last_Refresh | Timestamp per la registrazione dell'ultimo aggiornamento dei dati dello snapshot. |
| Error | Qualsiasi errore incontrato durante l'ultimo tentativo di aggiornamento. |
| Type | Tipo di aggiornamento effettuato ('COMPLETE', 'FAST' o 'FORCE'). |
| Next | Funzione di data utilizzata per determinare la data del prossimo aggiornamento. |
| Start_With | Funzione di data utilizzata per determinare le date di avvio e del prossimo aggiornamento. |
| Query | Query utilizzata come base per lo snapshot. |
| Refresh_Group | Nome del gruppo di aggiornamento a cui lo snapshot appartiene. |
| Update_Trig | In ORACLE8 questa colonna è sempre NULL ed è presente per compatibilità con le precedenti versioni di ORACLE. |
| Update_Log | Nome della tabella che registra le modifiche apportate a uno snapshot aggiornabile. |
| Refresh_Method | Valori utilizzati per pilotare un aggiornamento rapido dello snapshot. |
| FR_Operations | Stato delle operazioni di aggiornamento rapido ('REGENERATE' or 'VALID'). |
| CR_Operations | Stato delle operazioni di aggiornamento completo ('REGENERATE' or 'VALID'). |
| Master_Rollback_Seg | Segmento di rollback da utilizzare durante la popolazione dello snapshot e le operazioni di aggiornamento. |

La tabella base locale per tale vista è la colonna Table_Name in USER_SNAPSHOTS. La tabella utilizzata dallo snapshot nel database remoto è definita dalle colonne Master_Owner e Master.

Per determinare quali link di database sono utilizzati dai vari snapshot occorre effettuare una query sulla colonna Master_Link, come nell'esempio seguente.

```
select Master_Link
  from USER_SNAPSHOTS;
```

I nomi dei link di database restituiti da questa query possono essere utilizzati come valori in input per query nei confronti di USER_DB_LINKS. Tale query visualizza tutte le informazioni sui link di database utilizzati negli snapshot.

```
select *
  from USER_DB_LINKS
 where DB_Link in
       (select Master_Link
        from USER_SNAPSHOTS);
```

Nel Capitolo 28 sono descritte ulteriori query, utili per la gestione degli snapshot.

Le viste ALL_SNAPSHOTS e DBA_SNAPSHOTS hanno, per quanto riguarda le colonne, la stessa definizione di USER_SNAPSHOTS. ALL_SNAPSHOTS può essere utilizzata per visualizzare informazioni su tutti gli snapshot a cui l'utente è in grado di accedere, a prescindere dal proprietario. La vista DBA_SNAPSHOTS elenca informazioni sugli snapshot di tutti gli utenti del database.

Due nuove viste relative agli snapshot, USER_REFRESH e USER_REFRESH_CHILDREN, visualizzano informazioni sui gruppi di aggiornamento. Per ulteriori informazioni al riguardo si rimanda al Capitolo 28.

Log di snapshot: USER_SNAPSHOT_LOGS

I log di snapshot possono essere utilizzati da snapshot semplici per determinare quale record nella tabella master deve essere aggiornato negli snapshot remoti di tale tabella. Le informazioni riguardanti i log di snapshot di un utente possono essere ottenute mediante una query sulla vista del dizionario di dati USER_SNAPSHOT_LOGS, descritta nella Tabella 32.15. Le restrizioni all'utilizzo dei log di snapshot sono descritte nel Capitolo 28.

Tabella 32.15 Colonne di USER_SNAPSHOT_LOGS.

| NOME COLONNA | DESCRIZIONE |
|-------------------|---|
| Log_Owner | Proprietario del log di snapshot. |
| Master | Nome della tabella base per cui sono registrate modifiche. |
| Log_Table | Il nome della tabella che conserva i log (consistenti in RowID o chiavi primarie e date di modifica). |
| Log_Trigger | Nome del trigger AFTER ROW sulla tabella base (Master) che inserisce record nella tabella di log (Log_Table). |
| ROWIDs | Flag che indica se lo snapshot è basato su ROWID. |
| Primary_Key | Flag che indica se lo snapshot è basato su chiavi primarie. |
| Filter_Columns | Colonne filtro, per gli snapshot che utilizzano sottoquery. |
| Current_Snapshots | La data dell'ultimo aggiornamento dello snapshot. |
| Snapshot_ID | Numero unico di identificazione per lo snapshot. |

Le query su USER_SNAPSHOT_LOGS vengono normalmente effettuate per scopi di manutenzione, ad esempio per determinare il nome del trigger utilizzato per creare i record di log.

Non esiste una versione “ALL” di questa vista. DBA_SNAPSHOT_LOGS ha, per quanto riguarda le colonne, la stessa definizione di USER_SNAPSHOT_LOGS, ma visualizza tutti i log di snapshot nel database.

32.8 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 dell’attività o eseguire elaborazioni complicate. I trigger sono descritti nel Capitolo 23. Le procedure, le funzioni e i package sono descritti nel Capitolo 24. Nei paragrafi seguenti sono descritte le query sul dizionario di dati necessarie per ottenere informazioni su trigger, procedure, package e funzioni.

Trigger: USER_TRIGGERS

È possibile eseguire una query su USER_TRIGGERS per visualizzare informazioni sui trigger di proprietà del conto. Tale vista, le cui colonne sono elencate nella Tabella 32.16, visualizza il tipo e il corpo del trigger.

Tabella 32.16 Colonne di USER_TRIGGERS.

| 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. |
| Table_Name | Nome della tabella per cui è definito il trigger. |
| Referencing_Names | Nomi utilizzati per fare riferimento a valori OLD e NEW nel trigger (quando impostati diversamente dai valori predefiniti per OLD e NEW). |
| When_Clause | Clausola when utilizzata per il trigger. |
| Status | Stato di ENABLED o DISABLED del trigger. |
| Description | Descrizione del trigger. |
| Trigger_Body | Testo del trigger. |

La query seguente visualizza nome, tipo ed evento per tutti i trigger della tabella REGISTRO.

```
select Trigger_Name, Trigger_Type, Triggering_Event
  from USER_TRIGGERS
 where Table_Owner = 'TALBOT'
   and Table_Name = 'REGISTRO';
```

La vista del dizionario di dati ALL_TRIGGER elenca i trigger per tutte le tabelle a cui l'utente ha accesso. DBA_TRIGGER elenca tutti i trigger nel database. Ambedue le viste contengono una colonna Owner aggiuntiva con il proprietario del trigger.

Una seconda vista del dizionario di dati relativa ai trigger, USER_TRIGGER_COLS, visualizza come le colonne vengono utilizzate da un trigger. La vista elenca il nome di ciascuna colonna su cui influisce un trigger e l'utilizzo del trigger stesso. Come avviene per USER_TRIGGER, sono disponibili versioni "ALL" e "DBA" anche per questa vista del dizionario di dati.

Procedure, funzioni e package: USER_SOURCE

Il codice sorgente per procedure, funzioni, package e corpi di package esistenti può essere ricavato con una query sulla vista del dizionario di dati USER_SOURCE. La colonna Type in USER_SOURCE identifica l'oggetto procedurale come 'PROCEDURE', 'FUNCTION', 'PACKAGE', 'PACKAGE BODY', 'TYPE' o 'TYPE BODY'. Ciascuna riga di codice viene memorizzata in un record separato di USER_SOURCE.

La selezione di informazioni dalla vista USER_SOURCE richiede una query simile a quella riportata nell'esempio seguente. In tale esempio viene selezionata la colonna Text, ordinata per numero di riga (Line). Il nome dell'oggetto (Name) e il suo tipo (Type) vengono utilizzati per definire l'oggetto di cui verrà visualizzato il codice sorgente. Nell'esempio seguente viene utilizzata la procedura NUOVO_LAVORATORE descritta nel Capitolo 24.

```
select Text
  from USER_SOURCE
 where Name = 'NUOVO_LAVORATORE'
   and Type = 'PROCEDURE'
 order by Line;
```

TEXT

```
procedure NUOVO_LAVORATORE
  (Nome_Persona IN varchar2)
AS
BEGIN
  insert into LAVORATORE
    (Nome, Eta, Alloggio)
  values
    (Nome_Persona, null, null);
END;
```

Come mostrato nell'esempio precedente, la vista USER_SOURCE contiene un record per ciascuna riga della procedura NUOVO_LAVORATORE. La sequenza delle righe viene registrata nella colonna Line; questa deve pertanto essere utilizzata nella clausola order by, come nell'esempio.

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 a cui l'utente è in grado di accedere, a prescindere dal proprietario. DBA_SOURCE elenca il codice sorgente per tutti gli utenti del database.

Errori di codice: USER_ERRORS Il comando SQLPLUS show errors verifica la vista del dizionario di dati USER_ERRORS alla ricerca degli errori associati al più recente tentativo di compilazione di un oggetto procedurale. Il comando show errors visualizza il numero di riga e colonna per ciascun errore, insieme al testo del messaggio relativo.

Per visualizzare gli errori associati con oggetti procedurali creati in precedenza è possibile effettuare una query diretta su USER_ERRORS. Ciò può rendersi necessario per visualizzare gli errori associati con corpi di package, in quanto la compilazione di un package che restituisca un errore può non visualizzare l'errore nel corpo del 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. Le colonne presenti nella vista USER_ERRORS sono elencate nella Tabella 32.17.

Tabella 32.17 Colonne di USER_ERRORS.

| NOME COLONNA | DESCRIZIONE |
|--------------|--|
| Name | Nome dell'oggetto procedurale. |
| Type | Tipo dell'oggetto ('PROCEDURE', 'FUNCTION', 'PACKAGE', 'PACKAGE BODY', 'TYPE', o 'TYPE BODY'). |
| Sequence | Numero di sequenza della riga da utilizzare nella clausola order by della query. |
| Line | Numero di riga all'interno del codice sorgente in cui interviene l'errore. |
| Position | Posizione all'interno della riga in cui interviene l'errore. |
| Text | Testo del messaggio d'errore. |

Nell'esempio seguente è descritto un modello di query su USER_ERRORS. Le query su tale vista dovrebbero comprendere sempre la colonna Sequence nella clausola order by.

Di questa vista sono disponibili anche le versioni "ALL" e "DBA", dove compare la colonna aggiuntiva Owner che contiene il proprietario dell'oggetto.

```
select Line, Position, Text
  from USER_ERRORS
 where Name = 'NUOVO_LAVORATORE'
```

```
and Type = 'PROCEDURE'
order by Sequence;
```

Dimensione del codice: USER_OBJECT_SIZE La quantità di spazio utilizzato nel tablespace SYSTEM per un oggetto procedurale può essere ricavata con una query sulla vista sul dizionario di dati USER_OBJECT_SIZE. Come nell'esempio seguente, le quattro aree dimensionali separate possono essere sommate per determinare lo spazio totale utilizzato nella tabella 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 = 'NUOVO_LAVORATORE'
       and Type = 'PROCEDURE';
```

Di questa vista esiste anche la versione "DBA". DBA_OBJECT_SIZE elenca le dimensioni di tutti gli oggetti contenuti nel database.

32.9 Allocazione e utilizzo dello spazio, compreso il partizionamento

È possibile effettuare una query sul dizionario di dati per determinare lo spazio disponibile e allocato per gli oggetti del database. Nei paragrafi seguenti viene descritta la definizione dei parametri predefiniti di memorizzazione per gli oggetti, della percentuale di utilizzo dello spazio, dello spazio libero disponibile e il modo in cui gli oggetti vengono fisicamente memorizzati. Per ulteriori informazioni sui metodi di memorizzazione di ORACLE si rimanda al Capitolo 19.

Tablespace: USER_TABLESPACES

È possibile effettuare una query sulla vista del dizionario di dati USER_TABLESPACES per determinare a quali tablespace sia consentito l'accesso e i parametri predefiniti di memorizzazione di ciascuno di essi. Tali parametri vengono utilizzati per ciascun oggetto memorizzato all'interno di tale tablespace, a meno che un comando create o alter specifichi per tale oggetto parametri di memorizzazione differenti. Le colonne correlate alla memorizzazione della vista USER_TABLESPACES, elencate nella Tabella 32.18, sono molto simili alle colonne correlate alla memorizzazione di USER_TABLES. Per ulteriori informazioni si rimanda al paragrafo dedicato alla memorizzazione della guida alfabetica di riferimento del Capitolo 37.

Di questa vista non esiste una versione "ALL". DBA_TABLESPACES visualizza i parametri di memorizzazione per tutti i tablespace.

Tabella 32.18 Colonne di USER_TABLESPACES.

| NOME COLONNA | DESCRIZIONE |
|-----------------|--|
| Tablespace_Name | Nome del tablespace. |
| Initial_Extent | Parametro INITIAL predefinito per gli oggetti del tablespace. |
| Next_Extent | Parametro NEXT predefinito per gli oggetti del tablespace. |
| Min_Exts | Parametro MINEXTENTS predefinito per gli oggetti del tablespace. |
| Max_Exts | Parametro MAXEXTENTS predefinito per gli oggetti del tablespace. |
| Pct_Increase | Parametro PCTINCREASE predefinito per gli oggetti del tablespace. |
| Min_Extlen | Dimensioni predefinite dell'estensione minima per gli oggetti del tablespace. |
| Status | Stato del tablespace ('ONLINE', 'OFFLINE', 'INVALID', 'READ ONLY'). Un tablespace "invalid" è un tablespace eliminato. In questa vista i suoi record sono ancora visibili. |
| Contents | Flag che indica se il tablespace viene utilizzato 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 del tablespace. |

Percentuali 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. Nell'esempio seguente è riportato un modello di query su USER_TS_QUOTAS.

```
select * from USER_TS_QUOTAS;
```

| TABLESPACE_NAME | BYTES | MAX_BYTES | BLOCKS | MAX_BLOCKS |
|-----------------|-------|-----------|--------|------------|
| USERS | 67584 | 0 | 33 | 0 |

È presente un record per ciascun Tablespace_Name. La colonna Bytes riporta il numero di byte allocati agli oggetti di proprietà dell'utente. Max_Bytes è il massimo numero di byte che l'utente può possedere in quel tablespace. Se non è disponibile una percentuale di quel tablespace, Max_Bytes visualizza il valore 0, come nell'esempio precedente. Le colonne Bytes e Max_Bytes vengono tradotte in blocchi di ORACLE, rispettivamente nelle colonne Blocks e Max_Blocks.

Di questa vista non esiste una versione "ALL". DBA_TS_QUOTAS visualizza le percentuali di memorizzazione per tutti gli utenti e per tutti i tablespace, e rappresenta un modo molto efficace per elencare l'utilizzo dello spazio in tutto il database.

Segmenti ed estensioni: USER_SEGMENTS e USER_EXTENTS

Come descritto nel Capitolo 19, lo spazio viene allocato agli oggetti (ad esempio tabelle, cluster e indici) sotto forma di *segmenti*. I segmenti sono la controparte fisica

degli oggetti logici creati nel database. Per visualizzare i parametri di memorizzazione correnti e lo spazio effettivamente utilizzato per i propri 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 tale vista sono elencate nella Tabella 32.19.

Tabella 32.19 Colonne di **USER_SEGMENTS**.

| NOME COLONNA | DESCRIZIONE |
|-----------------|---|
| Segment_Name | Nome del segmento. |
| Segment_Type | Tipo del segmento ('TABLE', 'CLUSTER', 'INDEX', 'ROLLBACK', 'DEFERRED ROLLBACK', 'TEMPORARY', 'CACH', 'INDEX PARTITION', 'TABLE PARTITION'). |
| Tablespace_Name | Nome del 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'estensione iniziale del segmento. |
| Next_Extent | Valore del parametro NEXT per il segmento. |
| Min_Extents | Numero minimo di estensioni nel segmento. |
| Max_Extents | 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. |
| Partition_Name | NULL se l'oggetto non è partizionato. Altrimenti, il nome della partizione del segmento. |

I segmenti consistono in sezioni contigue dette *estensioni*. Le estensioni che costituiscono i segmenti sono descritte in **USER_EXTENTS**. Nella vista **USER_EXTENTS** è possibile visualizzare le effettive dimensioni di ciascuna estensione all'interno del segmento. Ciò si rivela molto utile per tenere traccia dell'impatto delle modifiche apportate alle impostazioni di `next` e `pctincrease`. Oltre a `Segment_Name`, `Segment_Type` e `Tablespace_Name`, **USER_EXTENTS** contiene quattro nuove colonne: `Extent_ID` (per identificare l'estensione all'interno del segmento), `Bytes` (le dimensioni dell'estensione, in byte), `Blocks` (le dimensioni dell'estensione, 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'utilizzo di spazio da parte degli oggetti dei vari proprietari.

Sia DBA_SEGMENTS, sia DBA_EXTENTS prevedono una colonna aggiuntiva Owner. Per elencare tutti i proprietari che possiedono segmenti di un tablespace, è possibile effettuare una query basata sulla colonna Tablespace_Name in DBA_SEGMENTS ed elencare tutti i proprietari di segmenti in tale tablespace.

Partizioni

Le partizioni consentono di suddividere i dati di una singola tabella su più tabelle. Per esempi di partizioni e una descrizione delle opzioni di indicizzazione disponibili si rimanda al Capitolo 19.

Per verificare se una tabella è partizionata, occorre effettuare una query sulla vista del dizionario di dati USER_PART_TABLES, le cui colonne sono elencate per categoria nella Tabella 32.20.

Tabella 32.20 Colonne di USER_PART_TABLES.

| IDENTIFICAZIONE | CORRELATE ALLA MEMORIZZAZIONE |
|------------------------|-------------------------------|
| Table_Name | Def_Tablespace_Name |
| Partitioning_Type | Def_Pct_Free |
| Partition_Count | Def_Pct_Used |
| Partitioning_Key_Count | Def_Ini_Trans |
| DefLogging | Def_Max_Trans |
| | Def_Initial_Extent |
| | Def_Next_Extent |
| | Def_Min_Exts |
| | Def_Max_Exts |
| | Def_Pct_Increase |
| | Def_Freelists |
| | Def_Freelist_Groups |

Come si vede nella tabella, la maggior parte delle colonne di USER_PART_TABLES definisce i parametri predefiniti di memorizzazione 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. La vista USER_PART_TABLES visualizza inoltre 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). In ORACLE8 l'unico valore di Partitioning_Type valido è 'RANGE'.

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 sola riga per ciascuna partizione della tabella. Le colonne della vista USER_TAB_PARTITIONS sono elencate, per categoria, nella Tabella 32.21.

Tabella 32.21 Colonne di USER_TAB_PARTITIONS.

| IDENTIFICAZIONE | CORRELATE ALLA MEMORIZZAZIONE | CORRELATE ALLE STATISTICHE |
|--------------------|-------------------------------|----------------------------|
| Table_Name | Tablespace_Name | Num_Rows |
| Partition_Name | Pct_Free | Blocks |
| High_Value | Pct_Used | Empty_Blocks |
| High_Value_Length | Ini_Trans | Avg_Space |
| Partition_Position | Max_Trans | Chain_Cnt |
| Backed_Up | Initial_Extent | Avg_Row_Len |
| Logging | Next_Extent | Sample_Size |
| | Min_Exts | Last_Analyzed |
| | Max_Exts | |
| | Pct_Increase | |
| | Freelists | |
| | Freelist_Groups | |

USER_TAB_PARTITIONS contiene colonne che identificano la tabella a cui la partizione appartiene e visualizza i parametri di memorizzazione della partizione e le statistiche per la stessa. Le colonne “correlate alle statistiche” sono popolate quando la tabella viene analizzata (si veda il comando `analyze` nella guida alfabetica di riferimento del Capitolo 37). Le colonne di identificazione visualizzano 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 utilizzate per la chiave di partizione è possibile accedere tramite la vista del dizionario di dati USER_PART_KEY_COLUMNS, che contiene solo tre colonne.

| | |
|-----------------|---|
| Name | Nome della tabella o indice partizionati. |
| Column_Name | Nome della colonna che fa parte della chiave di partizione. |
| Column_Position | Posizione della colonna all’interno della chiave di partizione. |

Alle statistiche per le colonne di partizione è possibile accedere tramite la vista del dizionario di dati USER_PART_COL_STATISTICS.

Le colonne presenti nella vista USER_PART_COL_STATISTICS, elencate nella Tabella 32.22, riflettono fedelmente le colonne contenute in USER_TAB_COL_STATISTICS.

Tabella 32.22 Colonne di USER_PART_COL_STATISTICS.

| NOME COLONNA | DEFINIZIONE |
|----------------|---|
| Table_Name | Nome della tabella. |
| Partition_Name | Nome della partizione. |
| Column_Name | Nome della colonna. |
| Num_Distinct | Numero di distinti valori nella colonna. |
| Low_Value | Valore inferiore nella colonna. |
| High_Value | Valore superiore nella colonna. |
| Density | Densità della colonna. |
| Num_Nulls | Numerosi valori NULL nella colonna. |
| Num_Buckets | Numero di bucket utilizzati per l'istogramma della colonna durante l'analisi. |
| Sample_Size | Dimensioni del campione utilizzato per analizzare la colonna. |
| Last_Analyzed | Data più recente in cui la colonna è stata analizzata. |

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 di dati `USER_PART_HISTOGRAMS`.

Le colonne contenute in tale vista sono `Table_Name`, `Partition_Name`, `Column_Name`, `Bucket_Number` ed `Endpoint_Value`.

Dato che gli indici possono essere partizionati, è disponibile una vista del dizionario di dati `USER_IND_PARTITIONS`. Le colonne contenute in `USER_IND_PARTITIONS` possono essere suddivise in tre categorie, elencate nella Tabella 32.23.

Tabella 32.23 Colonne di USER_IND_PARTITIONS.

| IDENTIFICAZIONE | CORRELATE ALLO SPAZIO | CORRELATE ALLE STATISTICHE |
|--------------------|-----------------------|----------------------------|
| Index_Name | Tablespace_Name | Blevel |
| Partition_Name | Ini_Trans | Leaf_Blocks |
| High_Value | Max_Trans | Distinct_Keys |
| High_Value_Length | Initial_Extent | Avg_Leaf_Blocks_Per_Key |
| Partition_Position | Next_Extent | Avg_Data_Blocks_Per_Key |
| Status | Min_Exts | Clustering_Factor |
| Logging | Max_Exts | Num_Rows |
| | Pct_Increase | Sample_Size |
| | Pct_Free | Last_Analyzed |
| | Freelists | |
| | Freelist_Groups | |

Le colonne presenti in USER_IND_PARTITIONS riproducono quelle contenute in USER_INDEXES, con alcune modifiche alle colonne di identificazione. Le colonne di identificazione per gli indici partizionati visualizzano il nome della partizione, il valore superiore della stessa, e la posizione della partizione all'interno della tabella. Le colonne “correlate alle statistiche” sono popolari quando la partizione viene analizzata (si veda il comando analyze nella guida alfabetica di riferimento del Capitolo 37). Le colonne “correlate allo spazio” descrivono l'allocazione dello spazio per l'indice. Per ulteriori informazioni sui parametri di memorizzazione si consulti il paragrafo dedicato alla memorizzazione della guida alfabetica di riferimento del Capitolo 37.

Spazio libero: USER_FREE_SPACE

Oltre a visualizzare lo spazio utilizzato, è possibile effettuare query sul dizionario di dati per sapere quanto spazio sia attualmente marcato come “libero”. La vista USER_FREE_SPACE elenca le estensioni libere in tutti i tablespace accessibili all'utente. La vista elenca per Tablespace_Name il File_ID, il Block_ID e il numero di file relativo del punto di inizio dell'estensione libera. Le dimensioni delle estensioni libere sono visualizzate sia in byte, sia in blocchi. DBA_FREE_SPACE viene frequentemente utilizzata dai DBA per controllare la quantità di spazio libero disponibile e il grado di frammentazione dello spazio medesimo.

32.10 Utenti e privilegi

Gli utenti e i relativi privilegi sono registrati all'interno del dizionario di dati. Nei paragrafi seguenti viene descritta l'esecuzione di query sul dizionario di dati per ottenere informazioni sugli account degli utenti, sui limiti di risorse e sui privilegi degli utenti stessi.

Utenti: USER_USERS

Per visualizzare informazioni sul proprio account è possibile effettuare una query su USER_USERS. In tale vista è possibile selezionare i propri Username, User_ID (un numero assegnato dal database), Default_Tablespace, Temporary_Tablespace e data Created (data in cui è stato creato l'account). La colonna Account_Status column in USER_USERS visualizza lo stato dell'account, precisando se questo è chiuso, aperto o scaduto. Se l'account è chiuso, la colonna Lock_Date visualizza la data di chiusura e la colonna Expiry_Date visualizza la data di scadenza.

ALL_USERS contiene solo le colonne Username, User_ID, e Created di USER_USERS, ma elenca tali informazioni per tutti gli account del database. La vista ALL_USERS si rivela utile quando si desidera sapere quali nomi utente siano disponibili (ad esempio durante l'esecuzione di comandi grant).

La vista DBA_USERS contiene tutte le colonne presenti in USER_USERS, più due colonne aggiuntive: Password (la password codificata per l'account) e Profile (il profilo risorse dell'utente). DBA_USERS elenca tali informazioni per tutti gli utenti del database.

Limiti delle risorse: USER_RESOURCE_LIMITS

In ORACLE i *profili* consentono di imporre limiti alle 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 nel paragrafo dedicato a create profile della guida alfabetica di riferimento del Capitolo 37. In ORACLE8 i profili vengono utilizzati per imporre misure addizionali di sicurezza, come date di scadenza per gli account e lunghezza minima delle password.

Per visualizzare i limiti attivi per la sessione corrente è possibile effettuare una query su USER_RESOURCE_LIMITS. In tale vista sono contenute le colonne elencate di seguito.

| | |
|---------------|--|
| Resource_Name | Nome della risorsa (ad 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.

Di questa vista non esistono versioni “ALL” o “DBA”. Essa è strettamente limitata alla sessione in corso dell'utente. Per visualizzare i costi associati a ciascuna risorsa disponibile è possibile effettuare una query sulla vista RESOURCE_COST. I DBA 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. Oltre alle colonne Grantee, Grantor e Owner, tale vista contiene le colonne Table_Name e 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 visualizza i record di USER_TAB_PRIVS in cui l'utente è concessionario (e pertanto non contiene una colonna Grantee). Dato che USER_TAB_PRIVS_MADE e USER_TAB_PRIVS_REC sono semplici sottoinsiemi di USER_TAB_PRIVS, è possibile emulare il loro funzionamento effettuando una

query su USER_TAB_PRIVS con un'adeguata clausola where per visualizzare il sottoinsieme desiderato.

Le versioni “ALL” sono disponibili per USER_TAB_PRIVS, USER_TAB_PRIVS_MADE e USER_TAB_PRIVS_RECD. Esse elencano gli oggetti per cui 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, per quanto riguarda le colonne, la stessa definizione 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. Ad esempio, è possibile concedere agli utenti la possibilità di aggiornare solo determinate colonne in una tabella. Per ulteriori informazioni si rimanda al Capitolo 18 e alla voce grant nella guida alfabetica di riferimento del Capitolo 37.

Le viste del dizionario di dati utilizzate per visualizzare privilegi di colonna sono praticamente identiche nella struttura alle viste relative ai privilegi di tabella descritte nel paragrafo precedente. L’unica modifica è l’aggiunta di una colonna Column_Name a ciascuna delle viste. 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.

Per tutte le viste relative ai privilegi di colonna esistono 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 su ‘YES’ o ‘NO’ per indicare se il privilegio è stato concesso with admin option). In questa vista vengono visualizzati tutti i privilegi di sistema concessi direttamente a un utente, come nell’esempio seguente. I privilegi di sistema concessi a un utente in grazia di un ruolo non sono visualizzati nello stesso elenco. Nell’esempio seguente all’utente Talbot è stato concesso il privilegio CREATE SESSION tramite un ruolo: tale privilegio non viene pertanto visualizzato.

```
select * from USER_SYS_PRIVS;
```

| USERNAME | PRIVILEGE | ADM |
|----------|------------------|-----|
| TALBOT | CREATE PROCEDURE | NO |
| TALBOT | CREATE TRIGGER | NO |

Non è disponibile la versione “ALL” di questa vista. Per visualizzare i privilegi di sistema concessi a tutti gli utenti del database occorre effettuare una query su DBA_SYS_PRIVS, che ha, per quanto riguarda le colonne, la stessa definizione di USER_SYS_PRIVS.

32.11 Ruoli

Oltre ai privilegi concessi direttamente agli utenti, è possibile raggruppare in ruoli serie ben determinate di privilegi. I ruoli possono essere concessi a utenti o ad altri ruoli e possono comprendere sia privilegi sugli oggetti, sia privilegi di sistema. Per ulteriori informazioni sull'utilizzo e la gestione dei ruoli si rimanda al Capitolo 18.

Per visualizzare i ruoli concessi a un utente, questi dovrà effettuare una query sulla vista del dizionario di dati USER_ROLE_PRIVS. In essa verrà visualizzato anche qualsiasi ruolo concesso al pubblico (PUBLIC). Le colonne disponibili per USER_ROLE_PRIVS sono elencate nella Tabella 32.24.

Tabella 32.24 Colonne di USER_ROLE_PRIVS.

| NOME COLONNA | DESCRIZIONE |
|--------------|--|
| Username | Nome utente (può essere 'PUBLIC'). |
| Granted_Role | Nome del ruolo concesso all'utente. |
| Admin_Option | Flag che indica se il ruolo è stato concesso with admin option ('YES' o 'NO'). |
| Default_Role | Flag che indica se il ruolo è il ruolo predefinito dell'utente ('YES' o 'NO'). |
| OS_Granted | Flag che indica se il sistema operativo verrà utilizzato per gestire i ruoli ('YES' o 'NO'). |

Per elencare tutti i ruoli disponibili nel database, è necessario disporre dell'autorità di DBA. Sarà quindi possibile effettuare una query su DBA_ROLES per elencare tutti i ruoli. La vista DBA_ROLE_PRIVS elenca le assegnazioni di detti ruoli a tutti gli utenti del database.

I ruoli possono ricevere tre differenti tipi di concessione e ciascuna di esse corrisponde a una differente vista del dizionario di 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 di dati mostrano solo i ruoli garantiti all'utente stesso.

Oltre a tali viste ne esistono altre due, ciascuna dotata di un'unica colonna, che elencano i privilegi e i ruoli concessi temporaneamente 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 concessi temporaneamente per la sessione. |

SESSION_PRIVS e SESSION_ROLES sono disponibili per tutti gli utenti.

NOTA *Per informazioni sull'attivazione e la disabilitazione dei ruoli e sull'impostazione di ruoli predefiniti si rimanda al Capitolo 18.*

32.12 Revisioni

In qualità di utente senza autorità di DBA all'interno di un database ORACLE, non si è in grado di attivare le funzioni di auditing del database stesso. Se però per il database è stata abilitata la auditing, saranno presenti viste del dizionario di dati utilizzabili da chiunque per visualizzare la procedura di auditing.

Per le procedure di auditing sono disponibili numerose e differenti viste del dizionario di dati. La maggior parte di tali viste è basata su una singola tabella di procedura di auditing nel database (SYS.AUD\$). La più generica vista disponibile è USER_AUDIT_TRAIL, le cui colonne sono descritte nella Tabella 32.25. Dato che tale vista visualizza i record di auditing per numerosi differenti tipi di azione, molte delle colonne potranno non essere applicabili a tutte le righe. La versione "DBA" di questa vista, DBA_AUDIT_TRAIL, elenca tutte le voci della tabella di auditing. La vista USER_AUDIT_TRAIL elenca solo le voci inerenti l'utente.

Tabella 32.25 Colonne di USER_AUDIT_TRAIL. (*continua*)

| 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 auditing. |
| Owner | Proprietario dell'oggetto influenzato da un'azione (per l'auditing di azioni). |
| Obj_Name | Nome dell'oggetto influenzato da un'azione (per l'auditing 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. |

Tabella 32.25 Colonne di USER_AUDIT_TRAIL.

| NOME COLONNA | DESCRIZIONE |
|---------------|---|
| 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 individuati durante la sessione. |
| Comment_Text | Commento testuale sulla voce relativa alla procedura di auditing. |
| 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. |
| Object_Label | Etichetta associata all'oggetto (per Trusted ORACLE). |
| Session_Label | Etichetta associata alla sessione (per Trusted ORACLE). |

Come si vede nella Tabella 32.25, la gamma di possibilità di auditing disponibili è piuttosto vasta (per l'elenco completo si consulti il comando audit nella guida alfabetica di riferimento del Capitolo 37). A ciascun tipo di auditing è possibile accedere tramite la relativa vista del dizionario di dati. Di seguito sono elencate 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 impartiti dall'utente. |

Per ciascuna delle viste "USER" elencate precedentemente esiste una versione "DBA" che visualizza tutti i record di procedura di auditing che ricadono nella categoria della vista.

Per visualizzare le opzioni di auditing attualmente abilitate per i propri oggetti occorre effettuare una query su USER_OBJ_AUDIT_OPTS. In USER_OBJ_AUDIT_OPTS, per ciascun oggetto menzionato sono elencate le opzioni di auditing per ciascun comando che potrebbe essere eseguito sull'oggetto stesso (identificato per mezzo delle colonne Object_Name e Object_Type). I nomi delle colonne di USER_OBJ_AUDIT_OPTS corrispondono alle prime tre lettere del comando (ad esempio Alt per alter, Upd per update). Ciascuna colonna registra il comando come revisionato quando il comando stesso è completato con successo ('S'), non è completato ('U') o in entrambi i casi. L'opzione predefinita di auditing attivata per eventuali nuovi oggetti nel database può essere visualizzata tramite la vista ALL_DEF_AUDIT_OPTS, che utilizza per l'assegnazione dei nomi alle colonne le stesse convenzioni utilizzate da USER_OBJ_AUDIT_OPTS.

I comandi che possono essere controllati vengono memorizzati in una tabella di riferimento denominata AUDIT_ACTIONS. In questa tabella sono presenti 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 numerose altre viste di auditing che non prevedono versioni "USER", comprese DBA_AUDIT_EXISTS, DBA_PRIV_AUDIT_OPTS, DBA_STMT_AUDIT_OPTS e STMT_AUDIT_OPTION_MAP. Per ulteriori informazioni su tali viste riservate ai DBA si rimanda alla *ORACLE Server Administrator's Guide*.

32.13 Monitoraggio: le tavole V\$ o tavole di prestazioni dinamiche

Le viste che consentono il monitoraggio delle prestazioni dell'ambiente vengono dette *statistiche*. Le tavole statistiche di sistema, dette anche *tabelle di prestazioni dinamiche*, sono comunemente conosciute come tavole V\$ in quanto iniziano tutte con la lettera "V" seguita dal simbolo del dollaro (\$).

Le definizioni e l'utilizzo delle colonne all'interno delle viste di monitoraggio sono soggetti a modifiche a ogni versione rilasciata del database. L'utilizzo delle viste in sé è però piuttosto statico. Le descrizioni delle viste sono riportate nella Tabella 32.26. La corretta interpretazione dei risultati di apposite query sulle viste richiede normalmente la consultazione della *ORACLE Server Administrator's Guide*. Le tavole V\$ sono normalmente utilizzate solo dai DBA.

Tabella 32.26 Le viste di monitoraggio V\$. (*continua*)

| NOME VISTA | DESCRIZIONE |
|--------------|---|
| V\$ACCESS | I lock attualmente mantenuti sugli oggetti del database. |
| V\$BGPROCESS | Informazioni sui procedimenti in background del database. |
| V\$DATAFILE | Informazioni sui file dati del database. |

Tabella 32.26 Le viste di monitoraggio V\$.

| NOME VISTA | DESCRIZIONE |
|----------------|---|
| V\$DBFILE | Vista pre-ORACLE7 utilizzata per elencare i file dati. Utilizzare al suo posto V\$DATAFILE. |
| V\$FILESTAT | Statistiche di lettura/scrittura file per i file di dati. |
| V\$LATCH | Latch interni del database, per numero di latch. |
| V\$LATCHNAME | Tabella codici che mappa numeri di latch su nomi di latch. |
| V\$LOCK | Informazioni su lock e risorse non-DDL. |
| V\$LOG | Informazione sui redo log del database. |
| V\$LOGFILE | Informazioni sui file di redo log del database. |
| V\$PARAMETER | Valori correnti dei parametri del database. |
| V\$PROCESS | Processi in corso. |
| V\$ROLLNAME | Tabella codici che mappa numeri di segmenti di rollback su nomi di segmenti di rollback. |
| V\$ROLLSTAT | Statistiche per tutti i segmenti di rollback in linea, per numero di segmento di rollback. |
| V\$SESSION | Sessioni in corso. |
| V\$SESSTAT | Statistiche correnti per la sessione in corso. |
| V\$SGA | Dimensioni di ciascun componente della SGA. |
| V\$SYSSTAT | Statistiche correnti per l'intero database. |
| V\$THREAD | Informazioni sui thread di redo log. |
| V\$TRANSACTION | Informazioni sulle transazioni. |

32.14 Varie

Oltre alle viste del dizionario di dati descritte in precedenza nel presente capitolo, esistono numerose viste e tabelle miste che possono essere disponibili all'interno del dizionario di dati. Tali viste e tabelle comprendono le viste riservate ai DBA e la tabella utilizzata quando viene eseguito il comando `explain plan`. Nei paragrafi seguenti sono riportate brevi descrizioni per ciascuno dei tipi misti di viste.

CHAINED_ROWS

Quando una riga non si adatta più all'interno del blocco di dati in cui è memorizzato il relativo titolo di riga, può memorizzare i dati restanti in un blocco o una serie di blocchi differente. Una simile riga viene detta *concatenata*. Le righe concatenate possono causare basse 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 nel proprio schema la tabella `CHAINED_ROWS` occorre eseguire lo script `UTLCHAIN.SQL` (che normalmente si trova nella sottodirectory `/rdbms/admin` della directory principale di ORACLE).

Per popolare la tabella `CHAINED_ROWS` si utilizza la clausola `list chained rows into` del comando `analyze`, come nell'esempio seguente.

```
analyze LEDGER list chained rows into CHAINED_ROWS;
```

La tabella `CHAINED_ROWS` elenca `Owner_Name`, `Table_Name`, `Cluster_Name` (se la tabella si trova in un cluster), `Partition_Name` (se la tabella è partizionata), `Head_RowID` (RowID della riga) e una colonna `TimeStamp` che visualizza la data e l'ora più recenti in cui la tabella o il cluster sono stati analizzati. È possibile effettuare una query sulla tabella basata sui valori di `Head_RowID` in `CHAINED_ROWS`, come nell'esempio seguente.

```
select * from REGISTRO
where RowID in
  (select Head_RowID
   from CHAINED_ROWS
   where Table_Name = 'REGISTRO');
```

Se la riga concatenata è di lunghezza ridotta, può essere possibile rimuovere la concatenazione eliminando e reinserendo la riga.

PLAN_TABLE

Quando si mettono a punto le istruzioni SQL, è possibile determinare i passaggi che l'ottimizzatore dovrà effettuare per eseguire la query. Per visualizzare il percorso di query è necessario in primo luogo creare nel proprio schema una tabella di nome `PLAN_TABLE`.

Lo script utilizzato per creare questa tabella si chiama `UTLXPLAN.SQL` e si trova normalmente nella sottodirectory `/rdbms/admin` della directory principale di ORACLE.

Una volta creata nel proprio schema la tabella `PLAN_TABLE`, è possibile utilizzare il comando `explain plan`. Questo comando genera record nella tabella `PLAN_TABLE`, contrassegnati dal valore di `Statement_ID` specificato per la query di cui si desidera visualizzare la successione dei passaggi.

```
explain plan
set Statement_ID = 'MYTEST'
for
select * from REGISTRO
where Persona like 'S%';
```

Le colonne `ID` e `Parent_ID` di `PLAN_TABLE` stabiliscono la gerarchia dei passaggi (Operations) che l'ottimizzatore seguirà per eseguire la query. Per ulteriori informazioni sull'ottimizzatore di ORACLE e l'interpretazione dei record di `PLAN_TABLE` si rimanda al Capitolo 36.

Interdipendenze: USER_DEPENDENCIES e IDEPTREE

Gli oggetti all'interno dei database ORACLE possono essere interdipendenti. Ad esempio, una procedura memorizzata 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 24).

Due serie di viste del dizionario di dati aiutano a tenere traccia di tali interdipendenze. 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 in pieno le interdipendenze è necessario creare nel proprio schema oggetti ricorsivi di segnalazione delle dipendenze. Allo scopo occorre eseguire lo script UTLDTREE.SQL (che normalmente si trova nella sottodirectory /rdbms/admin della directory principale di ORACLE). Tale script crea due oggetti su cui è possibile effettuare una query: DEPTREE e IDEPTREE. I due oggetti contengono informazioni identiche, ma IDEPTREE è rientrato in base alla pseudocolonna Level ed è pertanto più facile da leggere e interpretare.

Viste riservate ai DBA

Dato che il presente capitolo è dedicato agli sviluppatori e agli utenti, le viste del dizionario di dati riservate ai DBA non sono discusse. Queste viste sono utilizzate per ottenere informazioni su transazioni distribuite, dispute di lock, segmenti di rollback e altre funzioni interne del database. Per ulteriori informazioni sull'utilizzo delle viste riservate ai DBA si rimanda alla *ORACLE Server Administrator's Guide*.

Trusted ORACLE

Gli utenti di Trusted ORACLE sono in grado di visualizzare due ulteriori viste del dizionario di dati, elencate di seguito.

| | |
|-----------------|--------------------------------|
| ALL_LABELS | Tutte le etichette di sistema. |
| ALL_MOUNTED_DBs | Tutti i database montati. |

Per ulteriori informazioni sull'utilizzo di tali viste si rimanda alla *Trusted ORACLE Server Administrator's Guide*.

Viste di caricamento diretto SQL*LOADER

Per gestire l'opzione di caricamento diretto all'interno di SQL*LOADER, ORACLE mette a disposizione numerose viste del dizionario di dati. Su tali viste si

effettuano query normalmente solo per la risoluzione di problemi di programmazione, dietro richiesta del personale di supporto ai clienti di ORACLE. L'opzione di caricamento diretto SQL*LOADER è descritta alla voce "SQLLOAD" della guida alfabetica di riferimento del Capitolo 37. Le viste del dizionario di dati che la supportano sono elencate nella Tabella 32.27. Per queste viste non vengono immessi commenti descrittivi nel dizionario di dati.

Tabella 32.27 Viste del dizionario di dati per l'opzione Direct Path di SQL*LOADER.

| NOME VISTA |
|------------------------|
| LOADER_COL_INFO |
| LOADER_CONSTRAINT_INFO |
| LOADER_FILE_TS |
| LOADER_PARAM_INFO |
| LOADER_TAB_INFO |
| LOADER_TRIGGER_INFO |

Viste NLS (National Language Support)

Tre viste del dizionario di 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 nel file INIT.ORA del database o per mezzo del comando alter session. Per ulteriori informazioni sulle impostazioni NLS si consulti il paragrafo dedicato al comando alter session nella guida alfabetica di riferimento del Capitolo 37. Per visualizzare le impostazioni NLS correnti per la sessione, l'istanza o il database occorre effettuare una query rispettivamente su NLS_SESSION_PARAMETERS, NLS_INSTANCE_PARAMETERS o NLS_DATABASE_PARAMETERS.

Librerie

In ORACLE8 le routine PL/SQL (Capitolo 22) sono in grado di eseguire programmi in C esterni. Per visualizzare le librerie di programmi in 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 il la libreria può essere o meno 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 Owner aggiuntive che indica il proprietario della libreria. Per ulteriori informazioni sulle librerie si consulti il paragrafo dedicato a create library nella guida alfabetica di riferimento del Capitolo 37.

- Parte quarta
- **Ottimizzazione del progetto**

• Capitolo 33

• **L'importanza del fattore umano**

- 33.1 **Comprensione dei compiti dell'applicazione**
- 33.2 **Comprensione dei dati**
- 33.3 **Il modello commerciale**
- 33.4 **Inserimento dei dati**
- 33.5 **Query e report**
- 33.6 **Conclusioni**

Nel Capitolo 2 si è discussa la necessità di realizzare applicazioni comprensibili che si possano adattare alle esigenze degli utenti e sono state date specifiche raccomandazioni su come affrontare con successo tale compito. Si consiglia pertanto di rileggere il Capitolo 2, in quanto numerosi concetti presentati in tale capitolo servono come base per quanto discusso nel capitolo presente.

Il presente 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 deve 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 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 si crede essere progettazione, si tratti, 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. Ciò viene ottenuto migliorando la velocità e l'efficienza nell'adempiere alle operazioni commerciali e anche rendendo l'ambiente in cui 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. Togliete 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 nel presente capitolo non sono intesi come una trattazione rigorosa del procedimento e gli strumenti che si utilizzano, 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.

33.1 Comprensione dei compiti dell'applicazione

Uno dei passi spesso trascurati nella realizzazione del software è quello di comprendere a fondo quale sia il lavoro dell'utente finale, ovvero i 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 allargano in una serie di domande secondarie, che comprendono 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, mentre ciò è 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 deve essere eseguito, o è una conseguenza della modalità di progettazioni delle applicazioni per computer? L'utilizzo 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 o addirittura idealismo: ma 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 tali compiti sino al termine 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 consentono a fare in modo che sia questa idea a guidare l'analisi e la creazione di un'applicazione, invece dell'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 maggiormente a una rappresentazione di come le persone in effetti pensano e lavorano, di quanto non lo facesse il vecchio buon approccio di tipo "una cosa per volta". La lezione da ciò 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 in realtà 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 è quello di comprendere semplicemente che ciò è necessario. In questo modo, l'approccio all'analisi di un'applicazione economica è vista da una prospettiva nuova, naïve e scettica.

Il primo passo nell'analisi procedurale consiste nel comprendere i compiti da svolgere. Cosa rappresenta un determinato gruppo e cosa fanno i suoi componenti, in realtà, per guadagnarsi da vivere? 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, pensa 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 nel passato, orientamento che ha condotto decine di aziende a tentare di commercializzare ciò che essere ritenevano essere il "vero" prodotto, l'elaborazione dei dati, con risultati disastrosi per la maggior parte di esse.

Da ciò l'importanza della semplicità e dello scetticismo nell'apprendere quanto relativo a un'applicazione commerciale. Spesso è necessario sfatare le opinioni favorite di qualcuno sulla natura degli affari trattati, per scoprire la vera essenza di tale natura. Si tratta di un processo, seppure a volte difficoltoso, utile.

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, ad esempio grazie al presente 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 nel linguaggio locale, consente di realizzare sistemi di qualità straordinariamente alta. Ciascun componente del gruppo supporterà e apporterà 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 deve essere costituita da una semplice frase dichiarativa composta da un numero di parole variante da tre a dieci, in prima persona, senza virgole e con il minimo possibile di aggettivi.

Noi vendiamo assicurazioni.

La frase non deve essere simile a quella 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 descrittiva 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 semplici 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 manager, 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

nocciole di verità in essa contenuto dovrà fare parte del procedimento di documentazione dei compiti.

La naïveté portata dal progetto dell'applicazione nella cerchia di persone che si sforzano di documentare i compiti da svolgere, consente al primo di porre domande decisamente scettiche e individuare (per valutarne l'utilità) ciò che potrebbe essere un semplice retaggio. Il progettista deve comunque essere consci che, in quanto progettista, appunto, 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 manager 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, e normalmente molto prima, dall'implementazione di una nuova applicazione informatica.

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 ai compiti di livello zero. I livelli sottostanti vengono numerati utilizzando punti aggiuntivi, come in 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 lire 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 a seconda del 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 li propri scopi.

Nozioni ricavate dal documento dei dati Dal documento dei dati è possibile ricavare numerose nozioni. In primo luogo, il documento dei dati essendo orientato ai compiti invece che ai dati, muterà probabilmente in modo sostanziale il modo di progettazione delle finestre per l'utente.

Il documento influisce 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, che 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 sbagliati vengano corretti e che al documento dei compiti vengano apportati aggiustamenti passo dopo passo sino a che questo sia completo.

33.2 Comprensione dei dati

Insieme alla scomposizione e alla descrizione dei compiti, nel documento dei compiti vengono descritte le risorse necessarie per ogni passaggio. Ciò viene descritto 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. Questo è il modo con cui vengono costruiti gli elenchi di elementi di dati per i progetti. Il neo in questo tipo di approccio, descritto per la prima volta nel Capitolo 2, 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 domandarsi 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'utilizzo che viene fatto del dato inserito, tale elemento deve 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 ovvii. Se ciò 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 deve 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 utilizzati regolarmente nelle query in italiano. I nomi devono pertanto essere descrittivi (evitando le abbreviazioni, a parte quelle molto comuni in ambito commerciale) e singolari (per ulteriori informazioni, si faccia riferimento al Capitolo 35).

A causa dell'intima relazione tra il nome della colonna e i dati in questa conte-nuti, i due nomi devono essere assegnati contemporaneamente. La scelta ragionata del nome di una colonna semplifica di gran lunga l'assegnazione 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.

I nomi e i cognomi venivano uniti assieme, ad esempio, nel registro di Talbot e nella tabella creata per memorizzarne i dati. Si noti, comunque, che la tabella Nome 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 Nome deve essere scomposta in almeno due categorie 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 deve essere valutata elemento per elemento. Una volta terminata, ai nuovi elementi creati, destinati a diventare colonne, deve 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 numerosi buoni testi sull'argomento e una vasta gamma di strumenti CASE in grado di accelerare il processo. Per questo motivo nel presente volume non viene suggerito alcun metodo particolare, in quanto raccomandando un metodo si rischia di metterne in ombra un altro. I risultati hanno all'incirca l'aspetto illustrato nella Figura 33.1.

Per ciascuna transazione inscindibile deve essere creato uno schema come quello mostrato nella figura, a cui dovrà essere assegnato, come etichetta, il numero del compito a cui si riferisce.

Nello schema sono compresi i nomi delle tabelle, le chiavi primarie ed estranee e le colonne principali. Ciascuna relazione normalizzata (le linee di connessione) deve possedere un nome descrittivo e con ciascuna tabella deve comparire un calcolo stimato delle righe e percentuali di transazioni.

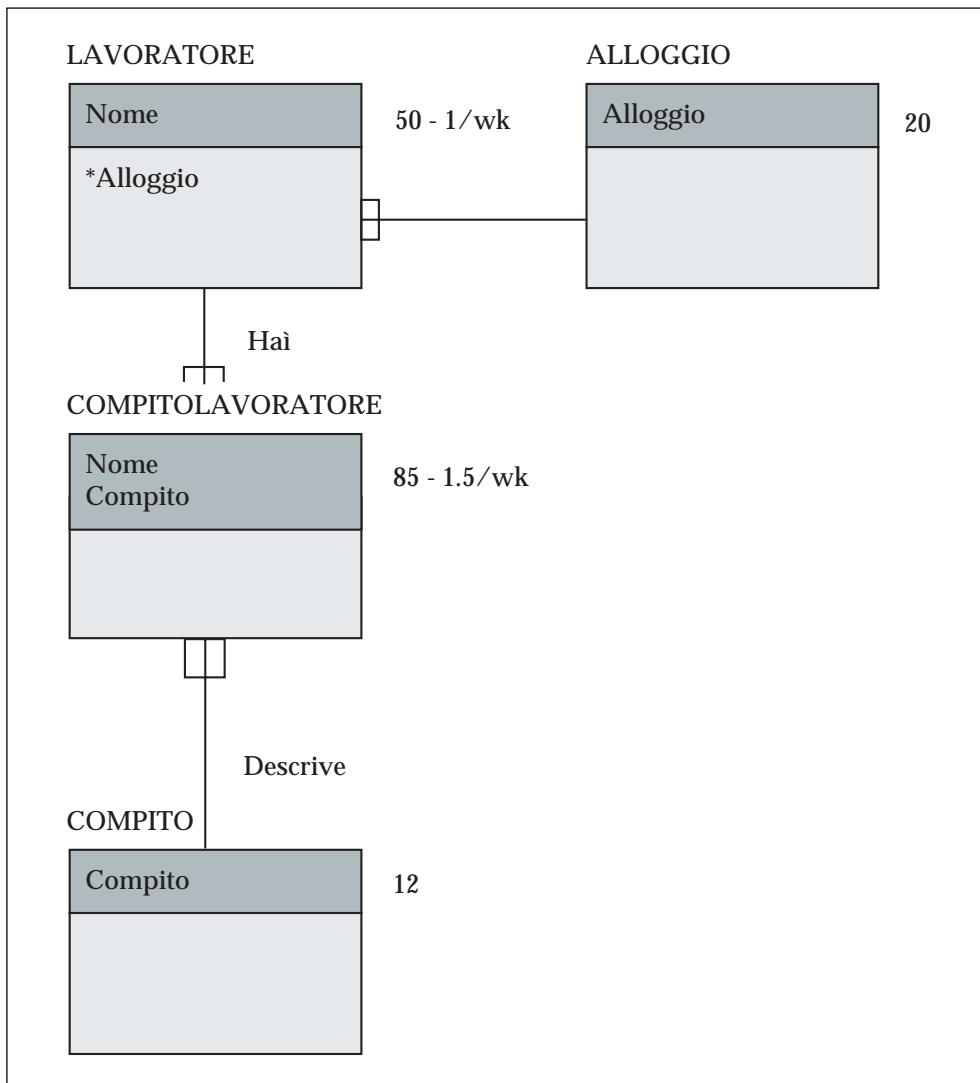


Figura 33.1 Un modello di dati per il compito di aggiungere un nuovo lavoratore.

Ogni schema 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.

33.3 Il modello commerciale

A questo punto, sia chi progetta l'applicazione, che 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 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.

33.4 Inserimento dei dati

La progettazione delle finestre di inserimento dei dati non deriva dal modello commerciale. Essa non si concentra sulle tabelle, ma 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 principale 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 come decisamente diverse da quelle tipiche orientate alla tabella sviluppate per la maggior parte delle 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 l'utente e macchina è fondamentale, le maschere di inserimento e le query devono essere coerentemente orientate ai compiti e descrittive, in italiano. 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.

33.5 Query e report

Se qualcosa 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), agli utenti finali viene assegnato 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, non avendo 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 consumino 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 quali tipi di query funzionano rapidamente e quali no. Il loro interesse potrà non essere rivolto specificamente ai cicli della CPU, ma essi desiderano sicuramente ottenere le proprie informazioni con rapidità. Ciò spinge gli utenti stessi ad apprendere le nozioni che rendono possibile la velocità e l'efficienza. Le stesse cognizioni riducono il consumo di cicli di calcolo. 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. Si ricordi la discussione sui costi della macchina paragonati ai costi dei dipendenti, riportata nel Capitolo 2. Praticamente ogniqualvolta si può spostare uno sforzo da una persona a una macchina, si risparmia denaro.

33.6 Conclusioni

Il progetto comprende problemi differenti che vanno dalle convenzioni di assegnazione dei nomi, dall'inserimento di dati, alla creazione di report. A seconda delle dimensioni e della natura dell'applicazione, problemi di volume delle transazioni, prestazioni e facilità di query costringono spesso alla violazione della Terza Forma Normale. Ciò non costituisce un peccato mortale, ma richiede la messa in opera di

determinati controlli per assicurare la coerenza e l'integrità dei dati. Tali problemi vengono discussi nel Capitolo 34.

A parte queste preoccupazioni, 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.

Lo scopo del presente capitolo non è quello di indirizzare verso una determinata serie di strumenti o di tecniche di schematizzazione, ma quello di descrivere la necessità di comprendere a fondo gli affari trattati, i dati e le esigenze degli utenti, per poter progettare e realizzare un'applicazione relazionale efficace. I dati possono essere inseriti in un database relazionale in modo banale. Ma le persone non organizzano i propri compiti basandosi sulla Terza Forma Normale dei propri dati. Un progetto che supporti adeguatamente il modo in cui una persona lavora deve andare oltre la visione dei dati. Tale progetto richiede una significativa riflessione e deve essere sviluppato con attenzione per poter essere implementato con successo in qualsiasi ambiente.

• Capitolo 34

• **Prestazioni e progettazione**

• 34.1 **Denormalizzazione e integrità dei dati**

• 34.2 **La tabella dei calcoli**

• 34.3 **Riepilogo**

In futuro nessuna delle principali applicazioni funzionerà nella terza forma normale. Questa affermazione suona probabilmente eretica in quanto detta a dispetto della moderna teologia relazionale, ma è vera. Forse, nel momento in cui le CPU diverranno più veloci e l'architettura a processi paralleli verrà meglio sfruttata, questa affermazione perderà di credibilità; più probabile sarà inoltre l'aumento della dimensione delle principali applicazioni. La richiesta di informazioni e di analisi continuerà probabilmente a ridurre la capacità da parte delle macchine di elaborare le richieste in modo totalmente normalizzato.

Ora, prima di lamentarsi dell'avvento di una nuova Inquisizione, quest'eresia deve essere spiegata.

Il tema della normalizzazione racchiude diversi aspetti. In questo capitolo non viene messo in discussione né il modello relazionale, che è contemporaneamente semplice ed elegante, né tantomeno il processo di normalizzazione, che rappresenta un eccellente e razionale metodo di analisi dei dati e delle loro relazioni fondamentali, ma semplicemente vengono sfataate alcune credenze non esatte.

- La normalizzazione consente una razionalizzazione completa dei dati.
- La normalizzazione rispecchia accuratamente il modo con cui l'uomo tratta i dati.
- I dati normalizzati costituiscono la rappresentazione migliore dei dati stessi.
- I dati memorizzati in modo non ridondante possono essere recuperati più velocemente dei dati memorizzati più volte.
- Le tabelle normalizzate rappresentano il modo migliore per memorizzare i dati.
- L'integrità referenziale richiede tabelle completamente normalizzate.

La scelta di un linguaggio “teologico” è intenzionale. Alcune persone parlano delle tecniche relazionali come se fossero leggi divine rivelate. La normalizzazione, ad esempio, rappresenta semplicemente un metodo per analizzare gli elementi che compongono i dati e le loro relazioni; il modello relazionale rappresenta una sovrastruttura teorica di supporto a questo processo. Assieme, i due forniscono un modo per rendere visivo l'ambiente dei dati, tuttavia non rappresentano il solo modo corretto o utile per vedere i dati. Infatti, in un gruppo complesso di relazioni anche la terza forma normale diventa presto insufficiente.

I moduli del livello superiore sono stati concepiti per affrontare queste relazioni più complesse, nonostante non vengano utilizzati spesso al di fuori del mondo accademico. I teorici concordano nel dire che essi sono insufficienti come completamento della realtà dei modelli (secondo il matematico Gödel, qualsiasi modello deve restare incompleto).

Nonostante ciò, l'adeguatezza del modello relazionale e la necessità della terza forma normale sono diventati, in certi ambienti, dogmi virtuali. Alcuni produttori (non ORACLE) hanno persino iniziato a imporre la terza forma normale nei propri dizionari di dati: non è consentito violarla, in caso contrario l'applicazione potrebbe non funzionare.

ORACLE adotta un approccio molto più pratico. Riconosce la genialità del modello relazionale, fino a obbedire alle regole di Codd, ma fornisce anche strumenti che consentono agli sviluppatori di utilizzare il proprio cervello e prendere le proprie decisioni riguardo alla normalizzazione, all'integrità referenziale, all'accesso al linguaggio procedurale, all'elaborazione dei dati non impostata a priori e altre simili tecniche "eretiche". Nel mondo reale, con dati e utenti reali e richieste concrete in termini di prestazioni e facilità di utilizzo, questa flessibilità è fondamentale per il successo. La normalizzazione è analisi, non progettazione. La progettazione comprende aspetti legati alle prestazioni, alla facilità di utilizzo, alla gestione e alla realizzazione in modo semplice delle operazioni commerciali, che non vengono considerati nella semplice normalizzazione.

34.1 Denormalizzazione e integrità dei dati

Quando si analizzano dei dati commerciali, la normalizzazione di questi almeno fino alla terza forma normale assicura che ciascuna colonna non chiave di ogni tabella dipenda soltanto dalla chiave primaria completa della tabella stessa. Se le relazioni tra i dati sono sufficientemente complesse, effettuare una normalizzazione a un livello più alto consente la comprensione profonda, anche se non completa, dei dati e delle relazioni tra i vari elementi da proteggere e gestire nel momento in cui vengono costruiti il database e l'applicazione.

Tuttavia, per applicazioni più grandi, o anche per semplici applicazioni in cui le operazioni non possono essere immediatamente ricondotte a tabelle normalizzate, una volta completata l'analisi il processo di progettazione può avere la necessità di denormalizzare alcune delle tabelle allo scopo di costruire un'applicazione che risponda alle esigenze, realizzi le operazioni dell'utente e riesca a completare il proprio processo nelle finestre di tempo disponibili. Esistono alcuni approcci utili in tal senso.

Tuttavia, qualsiasi raccomandazione atta a produrre risposte migliori deve necessariamente variare da applicazione ad applicazione e nel tempo, poiché migliorano i metodi di ottimizzazione delle query e sempre più potenza di calcolo della CPU viene trasferita ai dispositivi periferici del sistema e della rete. Effettuare il controllo delle prestazioni della propria applicazione all'interno del proprio sistema rappresenta l'unico modo per ottimizzare concretamente il proprio database.

Chiavi significative

Le tabelle poste nella terza forma normale del Capitolo 2 includono una delle tecniche presentate in questo paragrafo, nonostante precedentemente non sia stata citata. Una volta normalizzate, le tabelle e le colonne apparivano come in questa prima versione:

| LAVORATORE | COMPITOLAVORATORE | COMPITO | ALLOGGIO |
|------------|-------------------|-------------|-----------|
| Nome | Nome | Compito | Alloggio |
| Eta | Compito | Descrizione | NomeLungo |
| Alloggio | Capacita | | Direttore |
| | | | Indirizzo |

In un progetto più comune, soprattutto per grandi applicazioni con molte tabelle, potrebbero apparire più simili alla seguente versione:

| LAVORATORE | COMPITOLAVORATORE | COMPITO | ALLOGGIO |
|--------------|-------------------|-------------|------------|
| IDLavoratore | IDLavoratore | IDCompito | IDAlloggio |
| Nome | IDCompito | Compito | NomeCorto |
| Eta | Capacita | Descrizione | NomeLungo |
| IDAlloggio | | | Direttore |
| | | | Indirizzo |

La sostituzione di IDLavoratore al posto di Nome è probabilmente inevitabile. Troppo persone hanno nomi identici e ciò non consente al campo Nome di essere una chiave primaria univoca.

Tuttavia, questo non è necessariamente vero per le altre chiavi. In questo tipo di progetto, a ciascuna colonna ID viene solitamente assegnato un numero univoco sequenziale senza significato, infatti il suo unico scopo è quello di collegare una tabella a un'altra.

IDAlloggio, un numero, rappresenta una chiave esterna nella tabella LAVORATORE che punta a una sola riga della tabella ALLOGGIO. IDCmpito, un numero, rappresenta una chiave esterna nella tabella COMPITOLAVORATORE che punta a una riga della tabella COMPITO.

In questo sistema d'esempio, per avere informazioni riguardo alla mansione e alla collocazione di un lavoratore tutte e quattro queste tabelle devono essere combinate in una sola unione.

Tuttavia l'analisi delle operazioni mostra che molte delle query più comuni effettuano una ricerca del nome della mansione (Compito) e del nome dell'alloggio del lavoratore.

Le descrizioni aggiuntive relative al campo Compito e ai campi Indirizzo e Direttore riferiti all'alloggio vengono chiamate in causa solo raramente. Utilizzando la prima versione del progetto della tabella, dove Compito e Alloggio sono chiavi significative in quanto contengono realmente informazioni, non solamente un numero assegnato, solo le prime due tabelle necessitano di essere unite per eseguire le query più frequenti, come si può vedere dal listato a pagina seguente.

Tabella LAVORATORE

| NOME | ETA ALLOGGIO |
|----------------|---------------|
| Adah Talbot | 23 Papa King |
| Bart Sarjeant | 22 Cranmer |
| Dick Jones | 18 Rose Hill |
| Elbert Talbot | 43 Weitbrocht |
| Helen Brandt | 15 |
| Jed Hopkins | 33 Matts |
| John Pearson | 27 Rose Hill |
| Victoria Lynn | 32 Mullers |
| Wilfred Lowell | 67 |

Tabella COMPITOLAVORATORE

| NOME | COMPITO | CAPACITA |
|----------------|-------------|--------------|
| Adah Talbot | Operaio | Buono |
| Dick Jones | Fabbro | Eccellente |
| Elbert Talbot | Aratore | Lento |
| Helen Brandt | Trebbiatore | Molto veloce |
| John Pearson | Trebbiatore | |
| John Pearson | Taglialegna | Buono |
| John Pearson | Fabbro | Medio |
| Victoria Lynn | Fabbro | Preciso |
| Wilfred Lowell | Operaio | Medio |
| Wilfred Lowell | Aratore | Medio |

Invece di:

Tabella LAVORATORE

| NOME | ETA | IDALLOGGIO |
|----------------|-----|------------|
| Adah Talbot | 23 | 4 |
| Bart Sarjeant | 22 | 1 |
| Dick Jones | 18 | 5 |
| Elbert Talbot | 43 | 6 |
| Helen Brandt | 15 | |
| Jed Hopkins | 33 | 2 |
| John Pearson | 27 | 5 |
| Victoria Lynn | 32 | 3 |
| Wilfred Lowell | 67 | |

Tabella COMPITOLAVORATORE

| NOME | IDCOMPITO | CAPACITA |
|---------------|-----------|------------|
| Adah Talbot | 6 | Buono |
| Dick Jones | 4 | Eccellente |
| Elbert Talbot | 2 | Lento |

| | |
|----------------|----------------|
| Helen Brandt | 1 Molto veloce |
| John Pearson | 1 |
| John Pearson | 5 Buono |
| John Pearson | 4 Medio |
| Victoria Lynn | 4 Preciso |
| Wilfred Lowell | 6 Medio |
| Wilfred Lowell | 2 Medio |

A questo punto è necessario chiarire alcuni aspetti.

Per prima cosa le chiavi primarie significative devono essere, per definizione, uniche; esse devono assicurare la stessa integrità relazionale di una chiave numerica semplice.

In secondo luogo, i nomi delle chiavi primarie significative devono essere scelti con cura: possono avere lunghezza superiore a una parola e possono contenere spazi, purché siano brevi, descrittivi, mnemonici ed evitino i codici (eccetto quelli ampiamente noti e comuni in ambito commerciale). In questo modo è possibile utilizzarli nelle query con facilità e si riducono i comuni errori di inserimento quando li si utilizza per l'immissione dei dati.

In terzo luogo, la scelta di quali chiavi rendere significative deriva dall'analisi dei compiti da svolgere; la necessità di questa scelta è virtualmente indiscernibile dal processo di normalizzazione dei dati, poiché la normalizzazione può procedere con chiavi non significative (come nel secondo esempio) senza preoccuparsi dell'impatto di queste ultime sulle operazioni da svolgere.

È noto che nelle applicazioni principali, unioni che coinvolgono tre o più tabelle, anche se solo due di esse sono di grandi dimensioni, richiedono un numero di cicli di CPU significativo. Tuttavia, molte di queste unioni tra più tabelle possono essere evitate nel caso che un'analisi dei compiti da svolgere abbia portato alla comprensione di quali informazioni vengano richieste più spesso. Tutto ciò sembra implicare che il criterio su cui si misura un buon progetto sia rappresentato dalle prestazioni; in realtà questo è solo uno di vari criteri, che includono la semplicità, la comprensibilità, la possibilità di gestione e di controllo. L'utilizzo di chiavi significative può contribuire al raggiungimento di questi obiettivi.

In quarto luogo, nonostante la semplicità di una query possa essere migliorata attraverso le viste, facendo in modo che le quattro tabelle dell'esempio appaiano all'utente come un'unica tabella, questo metodo non risolve il problema delle prestazioni (nel caso vi sia un problema di questo tipo). Infatti in questo caso, forzando un'unione a quattro vie quando solo una o due tabelle risultano necessarie, non si farebbe altro che peggiorare a situazione.

In quinto luogo, l'utilizzo delle chiavi significative non è una vera e propria de-normalizzazione. L'integrità referenziale è ancora gestita allo stesso modo e le operazioni di update e delete su anomalie di forme al di sotto della terza non vengono realizzate attraverso questa tecnica. Alcuni puristi contestano l'utilizzo di chiavi significative, ma la purezza che essi inseguono ha spesso costi inaccessibili. Solo quando le CPU diventeranno infinitamente veloci potrà essere realizzato un tale livello di purezza. Inoltre, il problema reale non è con le chiavi significative, ma con le chiavi composte "intelligenti". Questo argomento viene trattato separatamente nel Capitolo 35.

Infine, se una query effettua la ricerca solamente nella colonna che è indicizzata, ORACLE può recuperare i dati solo tramite l'accesso all'indice, senza nemmeno accedere alla tabella sottostante, velocizzando in tal modo il recupero dei dati. Con una chiave significativa questo dato è utile, mentre con una chiave non significativa non lo è.

Nella costruzione di applicazioni relazionali di una certa entità, l'utilizzo delle chiavi significative è in grado di semplificare le relazioni tra le tabelle e rendere l'applicazione più veloce e intuitiva. Le chiavi significative richiedono analisi aggiuntive e un lavoro di progettazione nella fase di creazione dell'applicazione, ma ripagano velocemente degli sforzi fatti. Tuttavia, se l'applicazione è piccola e le risorse del computer più che sufficienti, la stessa semplicità e intuitività possono essere ottenuti attraverso l'utilizzo di viste che nascondano le unioni di tabelle all'utente finale.

Denormalizzazione reale

L'utilizzo di chiavi significative può non essere sufficiente. Le operazioni di select, insert, update, delete e i programmi batch e in linea possono risultare troppo lenti per applicazioni di grandi dimensioni, solitamente perché sono state unite tra di loro troppe tabelle.

Di nuovo, l'analisi delle operazioni da svolgere mostra le aree in cui l'informazione richiesta è forzata ad accedere a troppe tabelle contemporaneamente.

Una soluzione a questo problema consiste nel violare intenzionalmente la terza forma normale nel progetto delle tabelle inserendo dati ridondanti in una tabella quando essa non sia interamente dipendente dalla chiave primaria. Questa operazione non deve essere svolta con leggerezza, poiché il costo della denormalizzazione comporterebbe una maggiore difficoltà nella gestione del codice. Inoltre, gli argomenti citati nel Capitolo 2 riguardo al costo dell'attrezzatura risultano ancora validi. È spesso più conveniente acquistare più memoria e potenza di CPU che pagare costi di gestione e programmazione aggiuntivi. Dando per scontato che le prestazioni ne risulteranno influenzate in altri modi, la denormalizzazione rappresenta uno strumento efficiente.

Ad esempio, si supponga che, ogni volta che Talbot ha necessità di vedere il profilo di un impiegato, voglia anche conoscere il nome del direttore dell'area di lavoro dell'impiegato stesso. Viene quindi aggiunta questa colonna alla tabella LAVORATORE:

| LAVORATORE | COMPITOLAVORATORE | COMPITO | ALLOGGIO |
|------------|-------------------|-------------|-----------|
| Nome | Nome | Compito | Alloggio |
| Eta | Compito | Descrizione | NomeLungo |
| Alloggio | Capacita | | Direttore |
| Direttore | | | Indirizzo |

A questo punto i dati della tabella LAVORATORE sono ridondanti. Ad esempio, i dati di ciascun lavoratore dell'area di Cranmer ora contengono questa informazione in aggiunta al proprio nome e all'età:

| ALLOGGIO | DIRETTORE |
|----------|--------------|
| Cranmer | Thom Cranmer |

Se 50 dei lavoratori di Talbot sono nell'area di Cranmer, "Thom Cranmer" compare per 50 volte nella tabella LAVORATORE. Tuttavia, se l'applicazione è progettata in modo accurato, la terza forma normale è gestita logicamente, anche se viene violata nel progetto fisico delle tabelle. Questo può essere ottenuto continuando a gestire una tabella ALLOGGIO e imponendo due regole.

- Quando viene aggiunto un nuovo lavoratore alla tabella LAVORATORE, i dati della colonna Direttore devono provenire solo dalla tabella ALLOGGIO. A essa non può essere assegnata una chiave in modo indipendente. Se un nuovo lavoratore è collocato altrove e non è ancora nella tabella ALLOGGIO, è necessario che il suo alloggio venga aggiunto a quest'ultima, prima che il lavoratore venga aggiunto alla tabella LAVORATORE.
- Quando viene effettuata una modifica alla tabella ALLOGGIO, ad esempio il cambio di un direttore per Cranmer, ciascuna riga della tabella LAVORATORE che contiene Cranmer deve essere immediatamente aggiornata. Per tradurre in termini operativi, l'aggiornamento della tabella ALLOGGIO e delle colonne e righe della tabella LAVORATORE a essa correlate sono da considerarsi un'operazione inscindibile. Entrambe devono essere completate ed eseguite contemporaneamente. Tutto ciò rappresenta un compromesso tra migliori prestazioni e complessità della transazione.

Queste regole sono implementate da sempre. È inoltre possibile progettare un dizionario di dati in grado di gestire l'asserzione, in cui un'operazione inscindibile come quella descritta viene memorizzata ed eseguita automaticamente quando si apporta una modifica alla tabella ALLOGGIO, oppure la logica procedurale, che viene intrapresa dal dizionario se si verificano certe condizioni. PL/SQL e i trigger implementano queste regole all'interno di ORACLE.

Righe, colonne e volume

Ci si potrebbe ragionevolmente chiedere a che cosa può portare tutto ciò. Le query non risultano più lente a causa di questa ridondanza di dati nella tabella LAVORATORE? La risposta è no. Potrebbero anche risultare significativamente più veloci. Con solamente una o poche colonne di dati ridondanti, le probabilità sono a favore del fatto che le query, in particolare, risultano più veloci perché non vi è la necessità di unire le tabelle. Questo perché, quando una riga viene prelevata da una tabella correlata (ad esempio ALLOGGIO), tutte le colonne vengono riportate nella memoria della CPU e quelle non citate nella select vengono quindi scartate prima di effettuare l'unione.

Se solo una parte dei dati della tabella ALLOGGIO è mantenuta nella tabella LAVORATORE, viene prelevato ed elaborato dalla CPU, per l'operazione di select, un volume di dati minore. Oltre a ciò, se gli stessi dati vengono richiesti da più

query, la cache bufferizzata di ORACLE rende il tutto ancora più veloce. Le unioni e le tabelle multiple non vengono mai memorizzate nella cache; tuttavia gli aggiornamenti, soprattutto quelli apportati alla tabella ALLOGGIO, possono risultare più lenti, perché in tal caso necessitano di essere aggiornate anche le righe della tabella LAVORATORE.

Che cosa accade se la maggior parte, o persino tutti i dati di un tabella di riferimento vengono memorizzati in una tabella, ad esempio nel caso in cui Alloggio, NomeLungo, Direttore e Indirizzo si trovino nella tabella LAVORATORE (anche se si continua a utilizzare la tabella ALLOGGIO per l'integrità dei dati)? Quali sono le osservazioni al riguardo?

Una query che utilizzi un'unione di tabelle in cui solo la chiave esterna (alla tabella ALLOGGIO) risulti nella tabella LAVORATORE, mentre i dati di ALLOGGIO vengono mantenuti nella tabella omonima, causa nel complesso un minore spostamento di dati, verso la CPU dalle tabelle memorizzate sul disco. I dati nella riga di ALLOGGIO appartenenti a Cranmer, ad esempio, una volta recuperati, possono essere successivamente uniti a ciascuna riga correlata della tabella LAVORATORE nella memoria della CPU. In questo modo vengono spostati meno dati sul canale di I/O del disco rispetto al caso in cui si debbano recuperare gli stessi dati di Cranmer più volte per ciascun lavoratore dell'area di Cranmer, se tutte le colonne di ALLOGGIO risultano duplicate nella tabella LAVORATORE.

In sostanza vengono movimentati meno dati dal disco; in questo caso l'unione delle tabelle può risultare più veloce? Sorprendentemente, potrebbe non esserlo.

Dipendenza dalla memoria

I database relazionali sono solitamente legati alla memoria della CPU e ciò significa che la risorsa del sistema più limitante in termini di prestazioni è solitamente la memoria della CPU, non la velocità di accesso al disco, o la frequenza dell'I/O su disco per quanto riguarda il recupero iniziale dei dati. Ciò può sembrare contrario a quanto normalmente ci si aspetta e molti progettisti (incluso l'autore di questo libro) hanno speso ore di lavoro nell'organizzare accuratamente le tabelle su disco, evitando le discontinuità, minimizzando i movimenti delle testine e così via, tutto solo per ottenere un incremento marginale della produttività. Un sistema di database regolato in modo preciso, in cui tutte le risorse siano state bilanciate, collaudate e nuovamente bilanciate, può dipendere da fattori diversi dalla memoria della CPU, che tuttavia è solitamente il primo luogo a cui un sistema relazionale è legato. Questo legame è rappresentato dalla dimensione della memoria, anziché dalla velocità della CPU.

Quando la dimensione della memoria è troppo piccola, viene effettuata la suddivisione in pagine e le operazioni di registrazione, ordinamento e unione di tabelle presto coinvolgono non la memoria, l'I/O su disco, che è milioni di volte più lento. L'effetto che si ottiene è quello di frenare la CPU. Se alla macchina può essere aggiunta della memoria (ed è possibile avvantaggiarsene aumentando le dimensioni degli array o della SGA) vale la pena di affrontare la spesa fare. Anche in questo caso, effettuare la query su un'unica tabella con dati ridondanti può spesso risultare più veloce rispetto a un'unione.

In ogni caso, ORACLE fornisce delle funzionalità di monitoraggio che possono essere utilizzate dal DBA per determinare dove si trova il collo di bottiglia e attraverso le quali è possibile provare velocemente le query più utilizzate, con diversi progetti di tabelle fisiche alternative, utilizzando il set timing on in SQLPLUS o altri strumenti di controllo dell'impegno del disco forniti col sistema operativo. Quando si valutano le prestazioni, sono determinanti delle alternative di paragone. Ciò che intuitivamente sembra più veloce, spesso si rivela non esserlo.

La cassetta per gli attrezzi

Un'altra tecnica di progettazione pratica, che spesso fa gridare vendetta ai puristi citati precedentemente, è l'utilizzo della *tabella dei codici comuni*. Essa rappresenta l'equivalente del cassetto in cucina che raccoglie tutti gli arnesi, spago, vari strumenti e pezzi assortiti di cianfrusaglie a cui non è possibile trovare altra collocazione.

Ogni applicazione reale possiede bit di informazione e (quando non sono disponibili) codici racchiusi in un volume ridotto, spesso solamente in un'unica istanza. Possono esservi decine di queste istanze, tutte non correlate. Teoricamente queste non appartengono tutte allo stesso spazio, ma ciascuna di esse può avere tabelle separate. Nella pratica la gestione di tutto ciò può risultare difficile. La soluzione consiste nella realizzazione di un “cassetto per le cianfrusaglie”.

Questa struttura può racchiudere diverse colonne, ciascuna delle quali rappresenta una chiave primaria per solo una delle righe e NULL per tutte le altre. Può contenere colonne con nomi così generici che nessuno riuscirebbe a indovinarne il significato, con tipi di dati molto differenti a seconda della riga che si preleva.

Queste tabelle possono essere utili ed efficienti; tuttavia, è necessario seguire alcuni suggerimenti: in primo luogo, queste tabelle non dovrebbero solitamente essere visibili agli utenti. Possono far parte di una vista, ma l'utente finale non dovrebbe conoscerne l'esistenza. In secondo luogo, solitamente queste tabelle contengono informazioni utilizzate internamente, ad esempio per gestire un processo o per fornire una parte di informazione comune (quale l'indirizzo della propria società). In terzo luogo, la loro integrità deve essere assicurata da schermate e programmi progettati accuratamente, ai quali possa accedere solamente un gruppo limitato di sistemisti qualificati. Se qualche informazione contenuta in queste tabelle deve essere resa accessibile agli utenti, l'accesso deve essere governato rigidamente e probabilmente effettuato attraverso una vista che nasconde la forma reale della tabella.

Quali tecniche adottare

Queste tecniche, e altre simili, vengono talvolta tacciate di “violare la forma normale”; l'affermazione suggerisce che è stato fatto qualcosa di illogico o di errato, ma non è vero. Infatti, nella maggior parte degli esempi qui citati è stata conservata la forma normale nella logicità, se non nella progettazione fisica delle tabelle.

Il pericolo consiste nella violazione dell'integrità dei dati, in particolare dell'integrità referenziale: questo è il legame che collega i dati relazionali tra di loro e ne mantiene la coerenza. Se si fallisce in questo, si perdono informazioni, quasi sempre

in modo irrecuperabile. Questa rappresenta una preoccupazione legittima. Il modo in cui si progettano le tabelle, tuttavia, varia a seconda dei casi.

34.2 La tabella dei calcoli

La “ricerca tramite una tabella”, che risolve operazioni contenenti parti ben definite ricercando la risposta in una tabella anziché effettuare ogni volta i calcoli, rappresenta una tecnica ancora antecedente all’introduzione dell’elaborazione dei dati. L’utilizzo di una tabella di questo tipo è ovvio per strutture quali una tabella di numeri primi, ma può essere sfruttato anche in altre aree.

Molte operazioni di tipo finanziario, ad esempio, utilizzano due variabili x e y , e comprendono esponenti frazionari e moltiplicazioni e divisioni alquanto complesse; la parte relativa alla variabile x di ciascuna operazione è coinvolta solo in una delle quattro funzioni aritmetiche di base (quale la moltiplicazione).

Queste equazioni possono essere calcolate una volta, con la variabile x impostata a 1, per l’intero intervallo dei possibili valori di y . Il risultato per x e y viene quindi calcolato ricercando la y (che è indicizzata), recuperando il risultato precedentemente calcolato e memorizzato, e moltiplicandolo per la x corrente. Come con la denormalizzazione delle tabelle, il controllo delle prestazioni rappresenta il modo migliore per stabilire per quali calcoli questa tecnica risulta più produttiva.

34.3 Riepilogo

Probabilmente il dato più scoraggiante che si trae da tutto quanto si è detto sulla denormalizzazione, l’integrità referenziale, la dipendenza dalla memoria, il controllo delle prestazioni e così via è che il processo di progettazione si estende ben oltre la prima visione prodotta dall’analisi dei dati e delle operazioni da compiere. Sfortunatamente è proprio così. In modo particolare per le applicazioni di un certo rilievo, la parte di tempo di progettazione spesa nel calibrare le tabelle, nel controllo delle prestazioni e nella successiva ulteriore messa a punto può superare di molto il tempo speso per la progettazione iniziale. Non si tratta di semplici modifiche o ripensamenti: è necessario aspettarsi questa esigenza, preventivarla e pianificarla in modo rigoroso. È parte integrante del processo di progettazione quanto la determinazione delle chiavi primarie. L’applicazione più facile del mondo perde velocemente la propria popolarità se risulta troppo lenta.

Per costruire in modo sensibile un’applicazione è necessario bilanciare tre fattori principali: facilità di utilizzo, prestazioni e facilità di gestione. In un mondo perfetto questi tre aspetti risulterebbero tutti al livello massimo, ma in un mondo in cui i progetti hanno tempi di sviluppo limitati, le macchine hanno una potenza limitata e la facilità di utilizzo solitamente significa una complessità del programma aggiuntiva, è importante non favorire uno dei fattori alle spese di altri. Il vero messaggio di questo capitolo è comunicare la serietà dello sviluppo di un processo e l’importanza dell’attenzione alle forze in competizione esistenti.

• Capitolo 35

I dieci comandamenti della progettazione

- 35.1 **Verso la normalizzazione dei nomi degli oggetti**
- 35.2 **Sinonimi di nomi di oggetti**
- 35.3 **Chiavi intelligenti e valori di colonna**
- 35.4 **Comandamenti**

In questo capitolo vengono passati in rassegna in modo più completo alcuni degli aspetti riguardanti la progettazione di applicazioni che utilizzano database; vengono proposte strategie per implementare le idee presentate, vengono introdotte alcune nuove idee perché vengano analizzate dalla comunità relazionale e, infine, vengono redatti i “dieci comandamenti della progettazione”, meglio definiti come “dieci suggerimenti”. Questi ultimi hanno lo scopo di aiutare nella realizzazione di applicazioni produttive e di evitare che i progettisti possano cadere in profondi baratri (com’è già successo all’autore di questo libro).

35.1 Verso la normalizzazione dei nomi degli oggetti

Finora è stato detto abbastanza riguardo alle convenzioni di nomenclatura, prima nel Capitolo 2, poi nel Capitolo 33 e nel Capitolo 34. Queste idee sono state presentate in modo abbastanza informale: 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 Ultima_Data_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.

Nel seguito viene 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. Si faccia riferimento alla Figura 35.1.

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. Ad esempio, si considerino gli oggetti della Figura 35.2.

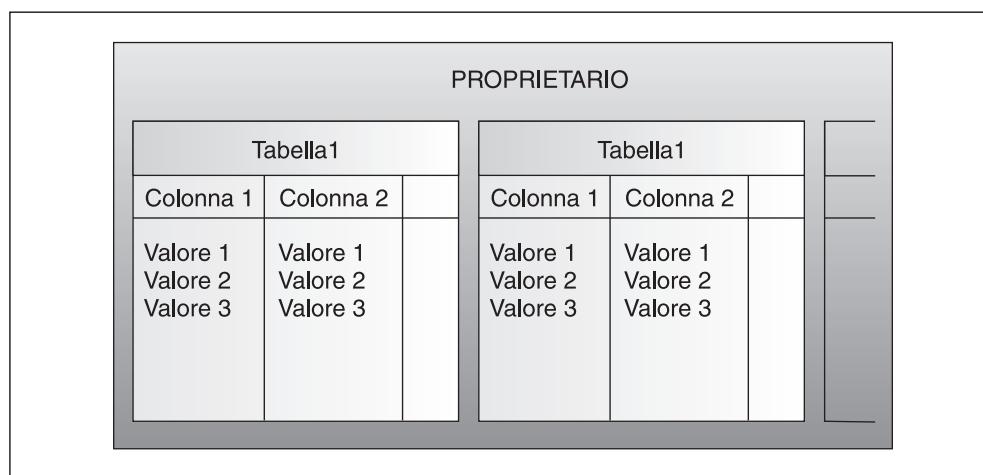


Figura 35.1 Gerarchia di database.

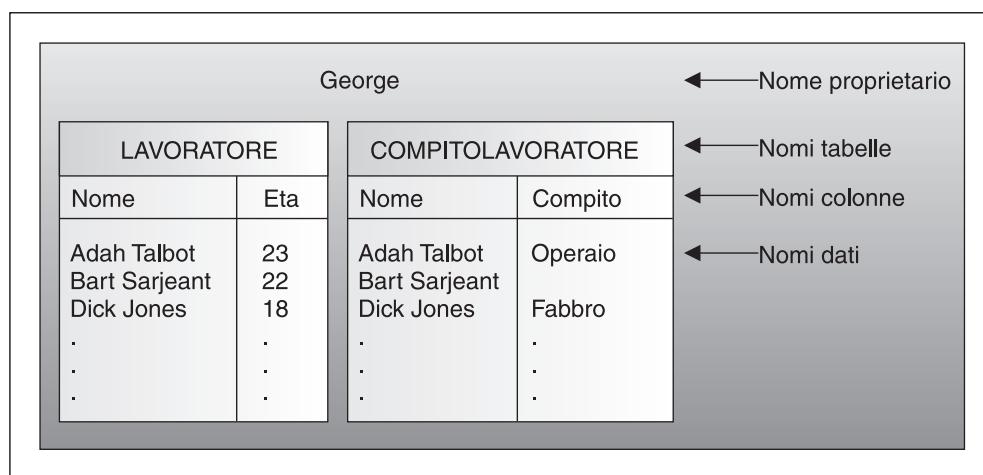


Figura 35.2 Esempio di database.

Il nome completo di una colonna mostra la propria eredità: GEORGE.LAVORATORE.Nome. Ciascun livello della gerarchia è separato dai livelli sottostanti e sovrastanti da un punto. I nomi devono essere unici all'interno del proprio livello superiore: LAVORATORE non può avere due colonne Nome.

Il proprietario George non può possedere due tabelle di nome LAVORATORE. Se Nome è una chiave primaria, non può avere due valori di dati identici. Tutto ciò è assolutamente ragionevole e logico.

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 LAVORATORE. 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.LAVORATORE. Se George combina una tabella LAVORATORE di sua proprietà con una di Dietrich, la colonna Nome in select deve essere identificata completamente, cioè occorre specificare Dietrich.LAVORATORE.Nome.

Non è coerente incorporare il nome del proprietario George nel nome di ciascuna delle sue tabelle, ad esempio, GEOLAVORATORE, GEOCOMPITOLAVORATORE 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*.

Tanto meno è logico o appropriato etichettare ciascun valore contenuto in una colonna con una parte del nome della colonna stessa, come in NomAdah Talbot, NomBart Sarjeant, NomDick Jones. Anche questa è una violazione dell'integrità del nome.

Ciò nonostante, molti progettisti di tabelle hanno adottato la tecnica di creare nomi di colonne che siano quanto più possibili univoci tra tutte le tabelle e per realizzare ciò inseriscono una parte del nome della tabella nel nome di ciascuna colonna, come in LAV_Nome, LAV_Eta, LAV_Compito. In alcuni casi questa è semplicemente una brutta abitudine acquisita dall'esperienza di lavoro con le tecnologie DBMS degli anni '70, quando era richiesta l'unicità di tutti i nomi dei campi nell'intero database. In altri casi, questo metodo viene adottato apparentemente nel tentativo di eliminare la necessità occasionale di nomi di tabelle nelle clausole select e where, in modo da produrre listati del tipo:

```
select LAV_Nome, LAV_Eta, LAV_Compito
  from LAVORATORE, COMPITOLAVORATORE
 where LAV_Nome = LC_Nome;
```

Anziché:

```
select LAVORATORE.Nome, Eta, Compito
  from LAVORATORE, COMPITOLAVORATORE
 where LAVORATORE.Nome = COMPITOLAVORATORE.Nome;
```

Questo approccio comporta problemi enormi. Il primo esempio non è molto esauriente. I prefissi richiedono chiavi aggiuntive, confondono il significato delle colonne e non si riesce a capire a quali tabelle fanno riferimento. Alcuni progettisti utilizzano come prefisso solo la prima lettera del nome della tabella.

In questo caso il problema dell'abbreviazione si fa immediatamente sentire ed è ancora più grave se vi sono più tabelle il cui nome inizia con la stessa lettera. Nell'esempio precedente, all'interno di una tecnica meno limitata, sono state adottate come abbreviazione delle due tabelle le lettere LAV e LC, ma il metodo di abbreviazione non viene definito da nessuna parte e la porzione del nome della colonna che definisce il significato di quest'ultima viene spinta verso destra.

Un'alternativa consiste nello specificare un prefisso, dato dall'abbreviazione della tabella, solo per le colonne il cui nome appare in più di una tabella:

```
select LAV_Nome, Eta, Compito  
      from LAVORATORE, COMPITOLAVORATORE  
     where LAV_Nome = LC_Nome;
```

Anche in questo caso la difficoltà è ovvia. Come fa l'utente a ricordare quali colonne hanno un prefisso e quali no? Che cosa accade a una colonna senza prefisso se viene aggiunta successivamente a un'altra tabella una colonna con lo stesso nome della prima? Viene cambiato il suo nome? Se sì, che cosa accade a tutti i report e le viste che si basano su di essa? Se il suo nome viene lasciato invariato, ma viene aggiunto un prefisso alla nuova colonna, come possono gli utenti ricordare quale colonna ha un prefisso e quale no?

Talvolta si fa notare che questo approccio consente di scrivere istruzioni SQL più concise poiché i nomi delle tabelle non devono essere specificati come chiavi nelle clausole select o where. Tuttavia, lo stesso livello di concisione può essere ottenuto utilizzando semplicemente gli alias delle tabelle, come mostrato di seguito:

```
select A.Name, Eta, Compito  
      from LAVORATORE A, COMPITOLAVORATORE B  
     where A.Nome = B.Nome;
```

Quest'ultimo metodo ha il vantaggio di includere l'abbreviazione scelta per ciascuna tabella e di renderla visibile nella clausola from.

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.

Chiavi esterne

Una difficoltà di questo approccio è la comparsa occasionale di una chiave esterna in una tabella in cui vi sia un'altra colonna con lo stesso nome che possiede la 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 (Figura 35.3).

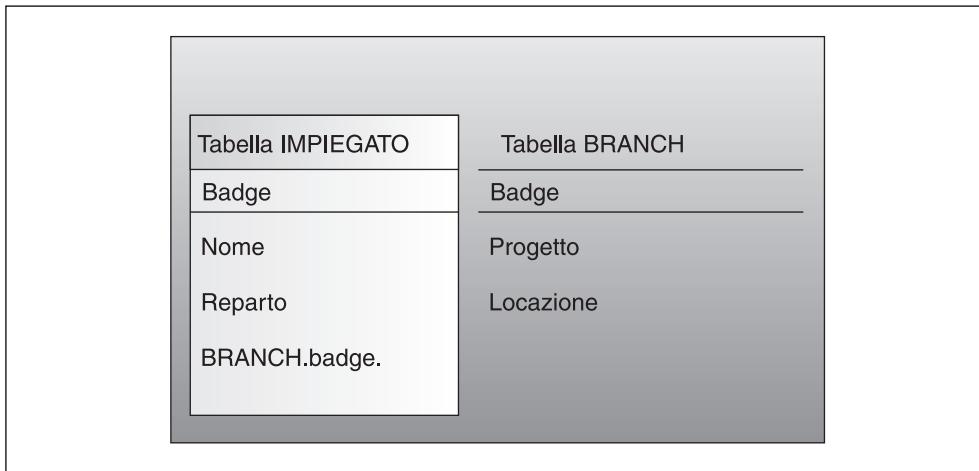


Figura 35.3 Chiave esterna con nome completo.

Se la query riguarda solo la tabella IMPIEGATO, BRANCH.Badge consente di identificare quella colonna. Se le due tabelle vengono unite, BRANCH.Badge fa riferimento in modo non ambiguo alla colonna Badge della tabella BRANCH. Per selezionare la stessa colonna, ma dalla tabella IMPIEGATO, il riferimento sarà IMPIEGATO.BRANCH.Badge. Questo metodo di identificazione delle chiavi esterne dovrebbe essere sufficientemente rigoroso, ma attualmente non è supportato. Viene qui proposto come idea da esplorare.

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 (ad esempio utilizzando il trattino di sottolineatura).
- Inventare un nome in cui venga incorporata 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 scegliere una di queste quattro vie.

Singolarità

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

Alcuni suggeriscono che i nomi delle tabelle debbano essere plurali, perché fanno riferimento a tutte le righe in esse contenute, perciò LAVORATORI. Anche i

nomi delle colonne fanno riferimento alle righe contenute, perciò anch'essi sarebbero plurali: ad esempio, colonna Nomi.

Altri sostengono che una tabella è in realtà un'insieme di righe ed è a una riga che i nomi delle tabelle e delle colonne fanno riferimento, perciò: LAVORATORE e Nome.

Un produttore propone che tutti i nomi delle tabelle dovrebbero essere plurali e tutti i nomi delle colonne singolari, perciò: LAVORATORI e Nome. Un altro propone che tutti i nomi delle tabelle vengano scritti in singolare, mentre lascia i nomi delle colonne a discrezione dell'utente. Esiste forse un po' di confusione tra progettisti e utenti, da tempo in sofferta attesa?

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 questi 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 il 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.

L'argomentazione per quanto riguarda i nomi delle tabelle è meno diretta. Esse contengono un insieme di righe. Un album fotografico contiene un insieme di molte fotografie. Un'agenda di indirizzi o di numeri di telefono contiene un gruppo di riferimenti a persone amiche o contatti di lavoro. Nessuno pensa a un'agenda con un unico indirizzo; ovviamente ne conterrà un certo numero e ciascuna istanza di questo insieme è rappresentata da un indirizzo. Ad esempio, può esserci un club automobilistico, un portafoglio di investimenti o un elenco dei migliori ristoranti.

D'altra parte, talvolta si parla di registro contabile, gruppo di single, elenco dei cibi preferiti. I gruppi vengono chiamati in entrambi i modi, sebbene la forma singolare sia più comune quando non vi sia preposizione. Si può dire "agenda telefonica", ma spesso si dice anche "agenda di indirizzi". Si tratta sempre di "tabella di lavoratori", non "tabella del lavoratore" (tra l'altro, "tabella dei lavoratori" suona come qualcosa di proprietà dei lavoratori, piuttosto che qualcosa che contiene informazioni su di essi).

Quindi, nonostante entrambe le soluzioni presentino vantaggi a seconda delle tabelle coinvolte, l'utilizzo del singolare in riferimento ai gruppi e agli insiemi è più comune nel linguaggio parlato.

Un secondo aspetto da considerare al riguardo chiama in causa la coerenza e l'utilità. 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. Tutto ciò è facile da utilizzare, da capire, da ricordare? Ovviamente no!

Brevità

Come si è già detto precedentemente, la chiarezza non dovrebbe mai essere sacrificata a beneficio della concisione, ma dati due nomi ugualmente significativi, mnemonicici e descrittivi, conviene sempre scegliere il più breve. Ad esempio, si supponga di assegnare un nome a una colonna che fa parte della descrizione della struttura di una società. Vi sono incluse colonne per il reparto, il dipartimento, il progetto e gli impiegati. La società in questione viene acquisita da un gruppo di controllo e il database della società diventerà il meccanismo per il servizio di informazioni del nuovo complesso. Ciò significa che risulta necessario un livello di organizzazione più elevato, per società. Come è possibile chiamarlo? Di seguito sono riportate alcune alternative:

- Ditta
- Impresa
- Attività
- Società
- Azienda

A seconda di come sono organizzati gli affari, ciascuno dei nomi precedenti può essere adatto (tutti sono stati utilizzati nei progetti delle tabelle). Ditta, tuttavia, ha una lunghezza minore rispetto ad Attività ed è significativo, mnemonico e descrittivo. Anche se non viene comunemente utilizzato quanto Società, ad esempio, è sicuramente più comune di Impresa, che è diventato piuttosto di moda, e viene facilmente appreso e ricordato una volta utilizzato.

Un altro esempio è rappresentato dalla scelta del nome per la tabella e la colonna ALLOGGIO. Potrebbe essere uno dei seguenti:

- Sistemazione
- Domicilio
- Residenza
- Collocazione
- Dimora
- Casa

Poiché Casa ha lunghezza pari a un terzo rispetto a Sistemazione e appena la metà di Alloggio, potrebbe essere la scelta migliore.

La brevità salvaguardia la struttura delle chiavi e rende concisi e facilmente comprensibili i nomi degli oggetti, ma è meno importante della chiarezza.

Durante lo sviluppo delle applicazioni, conviene proporre nomi 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 come quello presentato? 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.

35.2 Sinonimi di nomi di oggetti

In ultimo, 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).

Un oggetto può spesso contenere nel proprio nome più parole, ad esempio, Citta_Sede_Ditta. Se viene proposto un nuovo nome, come Citta_Domicilio_Società, il dizionario dei sinonimi potrebbe analizzare le componenti e giudicarle, quindi, parola per parola, approvare la scelta, oppure dichiarare una violazione degli standard e suggerire una valida alternativa, oppure segnalare la parola o l'abbreviazione che non è stata riconosciuta. L'utilizzo di un nome non approvato nella creazione di un oggetto dovrebbe richiedere l'approvazione da un gruppo o persona che si occupa degli standard, oppure produrre una relazione per il DBA, in cui si specifica la violazione.

Tale dizionario dei sinonimi può 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 e i costruttori di tastiere dovrebbero essere incoraggiati a spostare il tasto in un punto più facile da raggiungere di quanto non lo sia ora (spesso si trova nella fila inferiore e richiede la contemporanea pressione del tasto MAIUSC, perciò molte persone non sono in grado di premerlo senza guardare la tastiera).

35.3 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 intelligenti o 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 potrebbe essere 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, in tal caso

una ‘I’ viene accodata al numero, o se si tratta di un elemento voluminoso, ad esempio viti, 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 tabelle) 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 (ad 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).

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.

35.4 Comandamenti

I maggiori problemi che si riscontrano in fase di progettazione, per quanto riguarda la produttività, sono stati introdotti e trattati principalmente nei Capitoli 2, 33, 34 e in questo, ma brevemente anche in altre parti del libro. Probabilmente è utile riassumere gli aspetti affrontati in un unico elenco, ed ecco così “I dieci comandamenti”. Com’è già stato detto all’inizio di questo capitolo, un titolo migliore sarebbe “Dieci suggerimenti”. Vengono presentati non con l’intento di dire al lettore ciò che dovrebbe fare, ma piuttosto dando 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 utilizza il trattino di sottolineatura, lo si faccia coerentemente, oppure non lo 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. Decomporre 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 36

• **Guida all'ottimizzatore di ORACLE**

- 36.1 **Quale ottimizzatore?**
- 36.2 **Operazioni di accesso alle tabelle**
- 36.3 **Operazioni che utilizzano indici**
- 36.4 **Operazioni che gestiscono insiemi di dati**
- 36.5 **Operazioni di unione**
- 36.6 **Visualizzazione del percorso di esecuzione**
- 36.7 **Operazioni varie**
- 36.8 **Riepilogo**

Nel modello relazionale la locazione fisica dei dati non è importante. Anche un ORACLE la locazione fisica dei dati dell'utente e l'operazione utilizzata per recuperarli non sono importanti, finché il database non necessita di recuperare tali dati. Se si interroga il database, è necessario prestare attenzione alle operazioni eseguite da ORACLE per recuperare e gestire i dati. Quanto meglio si comprende il percorso esecutivo utilizzato da ORACLE per eseguire la query, tanto meglio si riuscirà a gestire e regolare la query stessa.

In questo capitolo vengono analizzate le operazioni utilizzate da ORACLE per interrogare ed elaborare i dati, dal punto di vista dell'utente. Si comincia con le operazioni per accedere alle tabelle, seguite dalle operazioni per accedere agli indici, dalle unioni e dalle operazioni di altro tipo. Per ciascun tipo di operazione vengono fornite informazioni per quanto riguarda la messa a punto, in modo da aiutare l'utente a utilizzarla nel modo più efficiente ed efficace possibile.

Prima di iniziare a regolare le proprie query, è necessario decidere quale ottimizzatore utilizzare.

36.1 **Quale ottimizzatore?**

L'ottimizzatore di ORACLE presenta tre modalità principali di funzionamento: RULE, COST e CHOOSE. Per impostare l'obiettivo dell'ottimizzatore è sufficiente specificare RULE, COST o CHOOSE come valore del parametro OPTIMIZER_MODE nel file init.ora del proprio database.

È possibile riassegnare le operazioni di default dell'ottimizzatore a livello di query e di sessione, come viene spiegato successivamente in questo capitolo.

Impostando il parametro OPTIMIZER_MODE al valore RULE viene attivato l'*ottimizzatore basato sulle regole* (RBO). Esso valuta i percorsi esecutivi possibili e fa una stima dei percorsi esecutivi alternativi in base a una serie di regole di sintassi.

Impostando OPTIMIZER_MODE al valore COST, viene attivato l'*ottimizzatore basato sui costi* (CBO). Si può utilizzare il comando analyze 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. In base a queste informazioni statistiche, l'ottimizzatore basato sui costi valuta il costo dei percorsi esecutivi disponibili e seleziona il percorso che ha il costo relativo più basso. Se si utilizza il CBO è necessario eseguire abbastanza spesso il comando analyze, per fare in modo che le statistiche rispecchino accuratamente i dati presenti nel database.

Impostando per il parametro OPTIMIZER_MODE il valore CHOOSE viene attivato l'ottimizzatore basato sui costi solo se le tabelle sono state analizzate, in caso contrario risulta attivo l'ottimizzatore basato sulle regole. Se una query fa riferimento a tabelle che sono state analizzate e ad altre che non lo sono state, il CBO può decidere di eseguire un esame completo delle tabelle non ancora analizzate. Per evitare potenziali analisi di tabelle non pianificate è opportuno evitare di utilizzare l'opzione CHOOSE, optando per l'ottimizzatore RBO o per l'ottimizzatore CBO sull'intero database.

36.2 Operazioni di accesso alle tabelle

Esistono due operazioni per accedere direttamente alle righe di una tabella: la scansione completa della tabella stessa e l'accesso alla tabella basato sul RowID. Per ulteriori 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.

Accesso completo

Durante una 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.

Viene utilizzata una scansione completa della tabella quando nella query non vi è alcuna clausola where. Ad esempio, la query seguente seleziona tutte le righe dalla tabella LAVORATORE:

```
select *
  from LAVORATORE;
```

Per risolvere la query precedente, ORACLE esegue una scansione completa sulla tabella LAVORATORE. Se questa tabella è di piccole dimensioni l'operazione può risolversi in breve tempo, senza incidere sulle prestazioni. Tuttavia, nel momento in cui la tabella LAVORATORE aumenta di dimensione, aumentano anche i costi dell'operazione in termini di prestazioni. Se più utenti eseguono scansioni complete della tabella, i costi associati crescono ancora più velocemente.

Accesso per RowID

Per migliorare le prestazioni delle operazioni di accesso alle tabelle si possono utilizzare le operazioni di ORACLE che consentono di impiegare i valori della pseudocolonna RowID per accedere alle righe della tabella. 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.

Quando si conosce il RowID, è noto con precisione dove la riga è fisicamente localizzata. Comunque, l'utente non deve memorizzare i RowID delle proprie righe, ma utilizza gli indici per accedere all'informazione riguardante i RowID, come viene descritto nel paragrafo "Operazioni che utilizzano gli indici". Poiché gli indici consentono un rapido accesso ai valori di RowID, aiutano a migliorare le prestazioni delle query che fanno uso di colonne indicizzate.

Hint

All'interno di una query è possibile specificare *hint* (suggerimenti) che dirigano il CBO nell'elaborazione della query stessa. Per specificare un hint si utilizza la sintassi mostrata nell'esempio che segue. Si inserisca la stringa seguente immediatamente dopo la parola chiave select:

```
/*+
```

Successivamente, si specifichi l'hint, ad esempio:

```
FULL(LAVORATORE)
```

Per chiudere l'hint si utilizzi la stringa seguente:

```
*/
```

Gli hint richiedono di utilizzare la sintassi di ORACLE per specificare i commenti all'interno delle query, con l'aggiunta del "+" all'inizio dell'hint. Nel corso di questo capitolo vengono descritti gli hint rilevanti per ciascuna delle operazioni. Per quanto riguarda gli accessi alle tabelle vi sono due hint rilevanti: FULL e ROWID. L'opzione FULL comunica a ORACLE di eseguire una scansione completa della tabella specificata (TABLE ACCESS FULL), come mostrato nel listato seguente:

```
select /*+ FULL(LAVORATORE) */  
      from LAVORATORE  
     where Alloggio = 'ROSE HILL';
```

L'opzione ROWID comunica all'ottimizzatore di utilizzare, per l'accesso alle righe della tabella, un'operazione TABLE ACESS BY ROWID. In generale si dovrebbe utilizzare questo tipo di operazione ogni volta che si ha la necessità di restituire velocemente le righe all'utente e quando le tabelle hanno dimensioni elevate. Per utilizzare l'operazione TABLE ACESS BY ROWID è necessario conoscere i valori di RowID, o utilizzare un indice.

36.3 Operazioni che utilizzano indici

All'interno di ORACLE vi sono due tipi principali di indici: gli *indici unici*, nei quali ciascuna 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 utilizzate 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 ALLOGGIO:

```
create table ALLOGGIO (
    Alloggio      VARCHAR2(15) not null,
    NomeLungo     VARCHAR2(40),
    Direttore     VARCHAR2(25),
    Indirizzo     VARCHAR2(30)
);
```

Per default non esistono indici creati in base alla tabella ALLOGGIO. Tuttavia si può dire che la colonna Alloggio rappresenta la chiave primaria della tabella, cioè identifica in modo univoco ciascuna riga e ciascun attributo dipendente dal valore di Alloggio. È possibile creare un vincolo di chiave primaria (PRIMARY KEY) sulla tabella ALLOGGIO per forzare l'unicità della colonna Alloggio e per consentire alle chiavi esterne (FOREIGN KEY) di fare riferimento ai valori di Alloggio.

Ogni volta che viene creata una chiave primaria o un vincolo di unicità (UNIQUE), ORACLE crea un indice unico per forzare l'unicità del valore nella colonna. Il comando alter table mostrato nel listato seguente comprende una clausola using index per forzare il collocamento dell'indice nel tablespace INDEXES:

```
alter table ALLOGGIO
    add constraint ALLOGGIO_PK primary key (Alloggio)
    using index tablespace INDEXES;
```

Come definito dal comando alter table, viene creata sulla tabella ALLOGGIO una chiave primaria di nome ALLOGGIO_PK. L'indice che supporta la chiave primaria si chiama ALLOGGIO_PK.

Si possono creare manualmente indici su altre colonne della tabella ALLOGGIO. Ad esempio, se i valori nella colonna Direttore non fossero univoci, si potrebbe creare un indice non unico sulla colonna tramite il comando create index:

```
create index ALLOGGIO$DIRETTORE
    on ALLOGGIO(Direttore)
    tablespace INDEXES;
```

La tabella ALLOGGIO possiede ora due indici: un indice unico sulla colonna Alloggio e uno non unico sulla colonna Direttore. Per la risoluzione di una query si possono utilizzare uno o più indici, a seconda del modo in cui la query stessa viene scritta ed eseguita.

INDEX UNIQUE SCAN

Per poter utilizzare un indice durante una query, quest'ultima deve essere scritta per consentire l'utilizzo di un indice. Nella maggior parte dei casi si consente all'ottimizzatore di utilizzare un indice attraverso la clausola `where` della query. Ad esempio, la query seguente può utilizzare l'indice unico sulla colonna Alloggio:

```
select *
  from ALLOGGIO
 where Alloggio = 'ROSE HILL';
```

Internamente, l'esecuzione della query precedente viene suddivisa in due fasi. Per prima cosa si accede all'indice ALLOGGIO_PK tramite un'operazione INDEX UNIQUE SCAN. Dall'indice viene restituito il valore di RowID che trova corrispondenza con il valore 'Rose Hill' di Alloggio; quel valore di RowID viene quindi utilizzato per interrogare ALLOGGIO attraverso un'operazione di TABLE ACCESS BY ROWID.

Se il valore richiesto dalla query fosse contenuto dall'interno dell'indice, ORACLE non avrebbe necessità di utilizzare l'operazione di accesso tramite RowId; nel caso in cui i dati si trovassero nell'indice, quest'ultimo rappresenterebbe tutto ciò che serve per soddisfare la query. Poiché in questo caso la query aveva selezionato tutte le colonne dalla tabella ALLOGGIO e l'indice non conteneva tutte le colonne della tabella ALLOGGIO, è stato necessario ricorrere all'operazione di TABLE ACCESS BY ROWID.

INDEX RANGE SCAN

Quando si interroga il database in base a un intervallo di valori, o si effettua una query utilizzando un indice non unico, si utilizza un'operazione di INDEX RANGE SCAN per interrogare l'indice.

Si consideri nuovamente la tabella ALLOGGIO, con un indice unico sulla propria colonna ALLOGGIO. Una query del tipo:

```
select Alloggio
  from ALLOGGIO
 where Alloggio like 'M%';
```

restituisce tutti i valori di Alloggio che iniziano con 'M'. Poiché la clausola `where` si riferisce alla colonna Alloggio, per risolvere la query può essere utilizzato l'indice ALLOGGIO_PK della colonna Alloggio. Tuttavia, nella clausola `where` non viene specificato un valore unico, bensì un intervallo di valori; dunque l'accesso all'indice unico ALLOGGIO_PK viene effettuato tramite un'operazione di INDEX RANGE

SCAN. Poiché le operazioni di INDEX RANGE SCAN richiedono la lettura di più valori dall'indice, sono meno efficienti delle operazioni di INDEX UNIQUE SCAN.

Nell'esempio precedente, dalla query è stata selezionata solo la colonna Alloggio. Poiché i valori della colonna Alloggio sono memorizzati nell'indice ALLOGGIO_PK, che è in fase di analisi, non vi è necessità per il database di accedere direttamente alla tabella ALLOGGIO durante l'esecuzione della query. L'operazione di INDEX RANGE SCAN di ALLOGGIO_PK è l'unica richiesta per risolvere la query.

La colonna Direttore della tabella Alloggio possiede un indice non unico sui propri valori. Se nella clausola where della query si specifica una condizione di vincolo per i valori di Direttore, viene eseguita un'operazione di INDEX RANGE SCAN dell'indice di Direttore. Poiché l'indice ALLOGGIO\$DIRETTORE non è unico, il database non può eseguire un'operazione di INDEX UNIQUE SCAN su ALLOGGIO\$DIRETTORE, anche se, nella query, la colonna Direttore viene confrontata con un unico valore. Ad esempio, la query:

```
select Alloggio
  from ALLOGGIO
 where Direttore = 'THOM CRANMER';
```

può accedere all'indice ALLOGGIO\$DIRETTORE tramite un'operazione di INDEX RANGE SCAN, ma non con un'operazione di INDEX UNIQUE SCAN. Poiché la query precedente seleziona la colonna Alloggio dalla tabella e la colonna Alloggio non si trova nell'indice ALLOGGIO\$DIRETTORE, nella query dell'esempio l'operazione di INDEX RANGE SCAN deve essere seguita da TABLE ACCESS BY ROWID.

Quando si utilizzano gli indici

Poiché gli indici hanno un grande impatto sulle prestazioni delle query, è necessario fare attenzione alle condizioni in cui un li si utilizza. 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 uguale a un certo valore Nella tabella ALLOGGIO la colonna Direttore ha un indice non unico di nome ALLOGGIO\$DIRETTORE. Una query in cui la colonna Direttore viene imposta a un certo valore è in grado di utilizzare l'indice ALLOGGIO\$DIRETTORE. La query seguente imposta la colonna Direttore al valore 'TOM CRANMER':

```
select Alloggio
  from ALLOGGIO
 where Direttore = 'THOM CRANMER';
```

Poiché l'indice ALLOGGIO\$DIRETTORE non è unico, questa query può restituire più righe e per leggere questi dati viene sempre utilizzata un'operazione di INDEX RANGE SCAN.

L'esecuzione della query precedente coinvolge due operazioni: un INDEX RANGE SCAN di ALLOGGIO\$DIRETTORE (per ricavare i valori di RowID di ciascuna riga che presenta il valore 'THOM CRANMER' nella colonna Direttore), seguita da un'operazione di TABLE ACCESS BY ROWID della tabella ALLOGGIO (per recuperare i valori della colonna Alloggio).

Se una colonna possiede un indice unico e viene impostata una query che la rende uguale a un certo valore, anziché un'operazione INDEX RANGE SCAN, viene effettuata un'operazione INDEX UNIQUE SCAN.

Se si specifica un intervallo di valori per una colonna indicizzata Per utilizzare un indice non è necessario specificare valori espliciti. L'operazione di INDEX SCAN OPERATION può analizzare un indice per intervalli di valori. Nella query seguente viene interrogata la colonna Direttore della tabella ALLOGGIO per un certo intervallo di valori (quelli che iniziano con 'R').

```
select Alloggio
  from ALLOGGIO
 where Direttore like 'R%';
```

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 utilizza l'indice ALLOGGIO\$DIRETTORE:

```
select Alloggio
  from ALLOGGIO
 where Direttore like '%CRANMER';
```

Poiché il primo carattere della stringa utilizzata per il confronto dei valori è un carattere jolly, non può essere utilizzato l'indice per trovare velocemente i dati corrispondenti. In questo caso viene eseguita una scansione completa della tabella (operazione TABLE ACCESS FULL).

Se nessuna funzione viene eseguita sulla colonna specificata nella clausola where Si consideri la query seguente, che utilizza l'indice ALLOGGIO\$DIRETTORE:

```
select Alloggio
  from ALLOGGIO
 where Direttore = 'THOM CRANMER';
```

Come si procede nel caso in cui non si sa se i valori della colonna Direttore siano memorizzati in lettere maiuscole, minuscole, o miste? Si può scrivere la query come:

```
select Alloggio
  from ALLOGGIO
 where UPPER(Direttore) = 'THOM CRANMER';
```

La funzione UPPER cambia i valori di Direttore in lettere maiuscole prima di confrontarli con il valore 'THOM CRANMER'. Tuttavia, utilizzando questa funzione sulla colonna si impedisce all'ottimizzatore di utilizzare un indice sulla colonna stessa. La query precedente (che utilizza la funzione UPPER) eseguirà un'operazione di TABLE ACCESS FULL della colonna ALLOGGIO.

Se si concatenano due colonne o una stringa e una colonna, non vengono utilizzati gli indici su quelle colonne. L'indice memorizza il valore reale della colonna e qualsiasi modifica di quel valore impedisce all'ottimizzatore di utilizzarlo.

Se per la colonna indicizzata non viene utilizzato nessun controllo IS NULL o IS NOT NULL I valori NULL non vengono memorizzati negli indici. Quindi, la query seguente non utilizza un indice; non vi è modo in cui l'indice possa aiutare a risolvere la query.

```
select Alloggio  
      from ALLOGGIO  
     where Direttore is null;
```

Poiché Direttore è l'unica colonna con una condizione di limitazione nella query e, in questo caso, la condizione di limitazione è un controllo di valore NULL, l'indice ALLOGGIO\$DIRETTORE non viene utilizzato e la risoluzione viene effettuata tramite un'operazione di TABLE ACCESS FULL.

Che cosa accade se sulla colonna viene effettuato un controllo IS NOT NULL? Tutti i valori non-NULI 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 di TABLE ACCESS BY ROWID associate) per tutti i valori restituiti dall'indice. Quindi, la query seguente non utilizza un indice:

```
select Alloggio  
      from ALLOGGIO  
     where Direttore is not null;
```

Se vengono utilizzate condizioni di equivalenza Negli esempi dei paragrafi precedenti, il valore di Direttore viene impostato uguale a un valore:

```
select Alloggio  
      from ALLOGGIO  
     where Direttore = 'THOM CRANMER';
```

Nel caso in cui si vogliano selezionare tutti i record che non presentano il valore 'THOM CRANMER' nella colonna Direttore, si deve sostituire "=" con "!=" , come nella query seguente:

```
select Alloggio  
      from ALLOGGIO  
     where Direttore != 'THOM CRANMER';
```

Nella risoluzione di quest'ultima query, l'ottimizzatore non è in grado di utilizzare un indice. Gli indici vengono utilizzati quando i valori vengono impostati con una relazione di uguaglianza, cioè quando le condizioni di limitazione sono di uguaglianza, non di diseguaglianza.

Un altro esempio di diseguaglianza è rappresentato dalla clausola not in, utilizzata in una sottoquery. La query seguente seleziona tutti i valori dalla tabella LAVORATORE per i lavoratori che non operano in settori gestiti da Thom Cranmer:

```
select Nome
  from LAVORATORE
 where Alloggio not in
(select Alloggio
   from ALLOGGIO
  where Direttore = 'THOM CRANMER');
```

La query del listato precedente non può utilizzare un indice sulla colonna Alloggio della tabella LAVORATORE, poiché questa colonna non viene impostata come uguale a nessun valore; al contrario il valore LAVORATORE.Alloggio viene utilizzato con una clausola not in per limitare le righe che trovano corrispondenza con quelle restituite dalla sottoquery. Per utilizzare un indice è necessario impostare la colonna indicizzata come uguale a un valore. La query seguente, che utilizza una clausola in, può utilizzare un indice sulla colonna LAVORATORE.Alloggio:

```
select Nome
  from LAVORATORE
 where Alloggio in
(select Alloggio
   from ALLOGGIO
  where Direttore = 'THOM CRANMER');
```

Se la colonna principale di un indice a più colonne viene impostata come uguale a un valore Un indice può essere creato su un'unica colonna o su più colonne. Nel secondo caso l'indice viene utilizzato solo 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, quest'ultimo non viene utilizzato per risolverla.

Se viene utilizzata la funzione MAX o MIN Se si seleziona il valore MAX o MIN di una colonna indicizzata, l'ottimizzatore può utilizzare l'indice per trovare velocemente il valore massimo o minimo contenuto nella colonna. Ad esempio, la query seguente può usufruire dell'indice ALLOGGIO\$DIRETTORE:

```
select MIN(Direttore)
  from ALLOGGIO;
```

Se l'indice è selettivo Tutte le regole precedenti che stabiliscono se un indice può essere utilizzato considerano solo la sintassi della query che viene eseguita e la struttura dell'indice disponibile. Se si sta utilizzando il CBO, l'ottimizzatore può sfruttare la selettività dell'indice per giudicare se l'indice sia in grado di abbassare il costo dell'esecuzione della query.

Se l'indice è altamente selettivo, allora viene associato un numero basso di record a ciascun valore distinto della colonna. Ad esempio, se vi sono 100 record in una tabella 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 di INDEX RANGE SCAN e di TABLE ACCESS BY ROWID utilizzate per recuperare i dati possono richiedere un lavoro maggiore rispetto all'operazione di 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. A partire da ORACLE7.3 l'ottimizzatore è in grado di utilizzare istogrammi per esprimere giudizi riguardo alla distribuzione dei dati all'interno di una tabella. Ad esempio, se i valori assunti dai dati sono pesantemente asimmetrici così 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. Gli istogrammi risultano abilitati per default a partire da ORACLE7.3.

Combinazioni di output da scansioni di più indici

Per risolvere una singola query possono essere utilizzati più indici, o scansioni multiple dello stesso indice. Per la tabella ALLOGGIO sono disponibili due indici: l'indice unico ALLOGGIO_PK sulla colonna Alloggio e l'indice ALLOGGIO\$DIRETTORE sulla colonna Direttore. Nei paragrafi seguenti viene illustrato in che modo l'ottimizzatore integra l'output di più scansioni attraverso le operazioni AND-EQUAL e CONCATENATION.

AND-EQUAL di più indici Se in una query vengono specificate condizioni di limitazione per più colonne indicizzate, l'ottimizzatore può utilizzare più indici.

La query seguente specifica dei valori sia per la colonna Alloggio sia per la colonna Direttore della tabella ALLOGGIO:

```
select *
  from ALLOGGIO
 where Alloggio > 'M'
   and Direttore > 'M';
```

Nella clausola where della query vi sono due condizioni di limitazione separate. Ciascuna di esse corrisponde a un indice differente: la prima a ALLOGGIO_PK e la seconda a ALLOGGIO\$DIRETTORE. Nella risoluzione della query l'ottimizzatore può utilizzare entrambi gli indici. Ciascun indice viene letto in sequenza attraverso un'operazione di INDEX RANGE SCAN.

I RowID restituiti dalla scansione dell'indice ALLOGGIO_PK vengono confrontati con quelli restituiti dalla lettura dell'indice ALLOGGIO\$DIRETTORE. I RowID restituiti da entrambi gli indici vengono utilizzati durante la successiva operazione di TABLE ACCESS BY ROWID. Nella Figura 36.1 viene mostrato l'ordine in cui le operazioni si susseguono.

L'operazione AND-EQUAL, come viene mostrato nella Figura 36.1, 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.

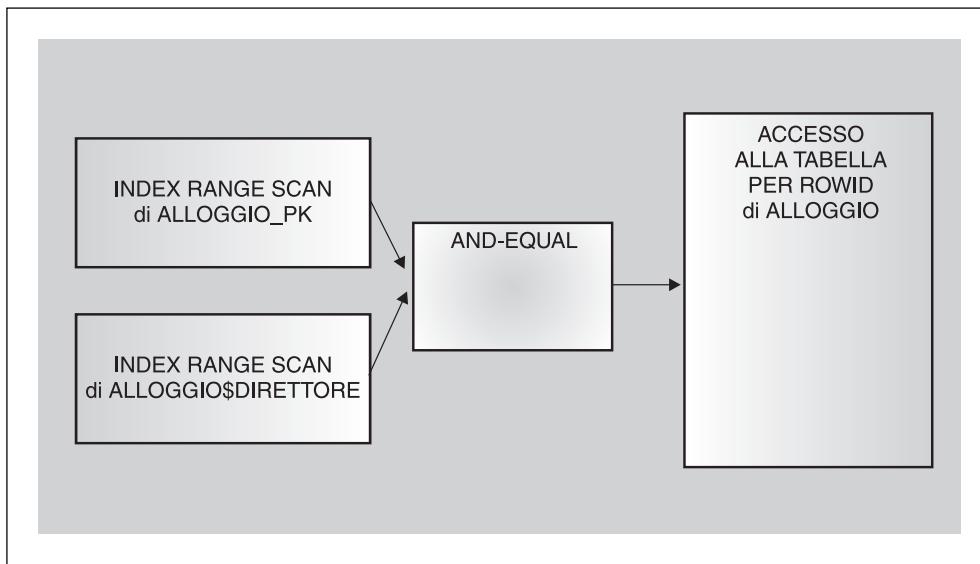


Figura 36.1 Ordine delle operazioni per AND-EQUAL.

Concatenazioni 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 concatenare i risultati di queste letture. Ad esempio, la query del listato seguente specifica due valori separati per quanto riguarda ALLOGGIO.Direttore:

```
select *
  from ALLOGGIO
 where Direttore in ('THOM CRANMER', 'KEN MULLER');
```

Poiché non si tratta di un intervallo di valori, un'unica operazione di 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.

L'ottimizzatore può interpretare la query precedente come se fossero state fornite due condizioni di limitazione separate, con una clausola or che rappresenta il legame tra le due:

```
select *
  from ALLOGGIO
 where Direttore = 'THOM CRANMER'
   or Direttore = 'KEN MULLER';
```

Quando si risolve la query, l'ottimizzatore può eseguire un'operazione di INDEX RANGE SCAN su ALLOGGIO\$DIRETTORE per ciascuna delle condizioni di limitazione. I RowID restituiti dalle scansioni dell'indice vengono utilizzati per accedere alle righe della tabella ALLOGGIO (per mezzo di operazioni di TABLE ACCESSIONE BY ROWID). Le righe restituite da ciascuna delle operazioni di TABLE

ACESS BY ROWID vengono combinati in un unico gruppo di righe attraverso l'operazione CONCATENATION. Nella Figura 36.2 viene mostrato l'ordine delle operazioni per la concatenazione delle due scansioni.

Hint

Sono disponibili diversi hint per dirigere l'ottimizzatore nella modalità di utilizzo degli indici. INDEX è il più comunemente utilizzato per quanto riguarda gli indici; esso comunica all'ottimizzatore di effettuare una scansione della tabella specificata basata sull'indice. Con questa opzione non è necessario specificare il nome dell'indice, anche se è possibile elencare indici specifici, se lo si desidera.

Ad esempio, nella query seguente viene utilizzata l'opzione INDEX per suggerire l'utilizzo di un indice relativo alla tabella ALLOGGIO durante la risoluzione della query:

```
select /*+ INDEX(Alloggio) */      Alloggio
      from ALLOGGIO
     where Direttore = 'THOM CRANMER';
```

Se si seguono le regole fornite precedentemente in questo paragrafo, questa query dovrebbe utilizzare l'indice senza che vi sia necessità dell'hint. Tuttavia, se l'indice non è selettivo e si sta utilizzando il CBO, l'ottimizzatore potrebbe scegliere di ignorare l'indice durante l'elaborazione dei dati. Se si sa che l'indice è selettivo per i valori dati, si può specificare l'opzione INDEX per forzare un percorso di accesso ai dati basato sull'indice.

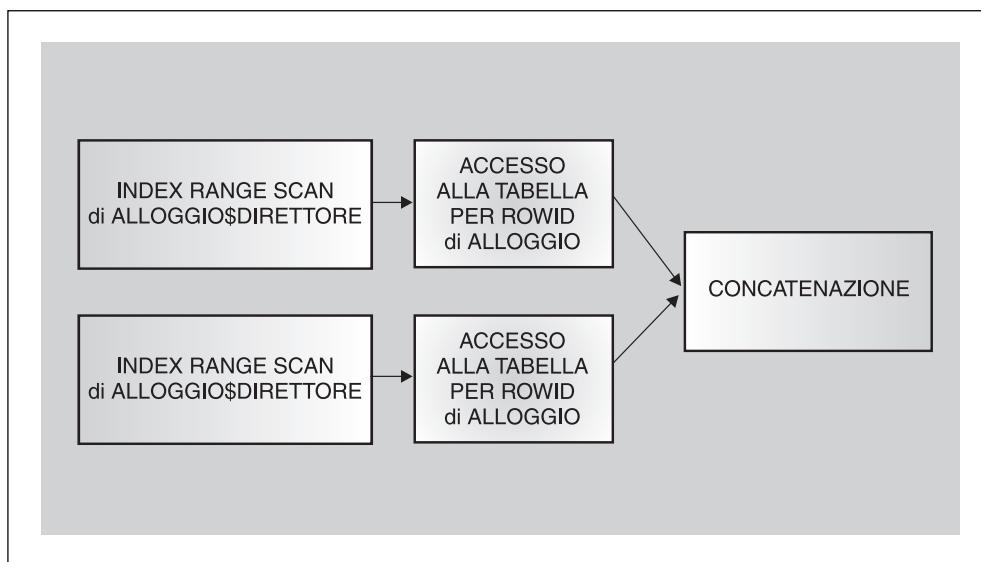


Figura 36.2 Ordine delle operazioni per CONCATENATION.

Nell'esempio che segue, all'interno dell'hint viene specificato l'indice ALLOGGIO\$DIRETTORE della tabella ALLOGGIO; questo indice viene coinvolto nella risoluzione della query.

```
select /*+ INDEX(Alloggio Alloggio$Direttore) */      Alloggio
      from ALLOGGIO
     where Direttore = 'THOM CRAMMER';
```

Se non si specifica un indice particolare 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 scandire più indici e unirli attraverso l'operazione AND-EQUAL descritta nel paragrafo precedente.

Esiste una seconda possibilità, INDEX_ASC, che funziona allo stesso modo di INDEX: questa opzione suggerisce una scansione in ordine ascendente per risolvere le query riguardanti tabelle specifiche. Una terza opzione, INDEX_DESC, comunica all'ottimizzatore di scandire l'indice in ordine discendente (dal valore più alto al valore più basso).

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 crea un indice concatenato, quale colonna stabilire come colonna principale dell'indice?

In generale l'ottimizzatore riesce a scandire più velocemente un unico indice concatenato che non a leggere e unire due indici separati. Quante più righe vengono restituite dalla lettura, tanto più probabile è che l'indice concatenato superi come prestazioni l'unione delle scansioni dei due indici. Se si aggiungono ulteriori colonne all'indice concatenato, risulta meno efficiente per scansioni di intervalli.

Per l'indice concatenato, quale colonna dovrebbe essere la 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 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 precedentemente), l'indice non viene chiamato in causa.

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 incrementa notevolmente 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.

La maggior parte delle applicazioni enfatizza i processi di transazione in linea rispetto ai processi batch; possono esservi 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 spesso rispetto al caso in cui

venga eseguita la scansione completa di una tabella. Quindi, nella fase di creazione dell'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 familiarità con le query eseguite nei confronti del 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.

36.4 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 vengono analizzate le operazioni che consentono di effettuare manipolazioni dei 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. Ad 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 di INDEX RANGE SCAN, la prima riga restituita dalla query passa i criteri impostati nelle condizioni di limitazione, quindi non vi è necessità di valutare il successivo record restituito prima di visualizzare quello già trovato. Al contrario, se viene eseguita un'operazione quale un ordinamento, i record non vengono immediatamente visualizzati. Durante operazioni di questo tipo, l'utente deve attendere che tutte le righe vengano elaborate. Quindi è necessario limitare il numero di questo tipo di operazioni per le query eseguite 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

Tre delle operazioni interne di ORACLE ordinano le righe senza raggrupparle. La prima è SORT ORDER BY, utilizzata quando nella query è presente la clausola order by. Ad esempio, si supponga che sia interrogata la tabella LAVORATORE. Questa tabella possiede tre colonne: Nome, Eta e Alloggio. La colonna Eta della tabella LAVORATORE non è indicizzata. La query seguente seleziona tutti i record dalla tabella LAVORATORE, ordinati secondo la colonna Eta:

```
select Nome, Eta
  from LAVORATORE
 order by Eta;
```

Quando questa query viene eseguita, l'ottimizzatore recupera i dati dalla tabella LAVORATORE tramite un'operazione di TABLE ACCESS FULL (poiché non vi sono condizioni di limitazione per la query, tutte le righe vengono restituite). I record recuperati non vengono immediatamente visualizzati all'utente; 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 tutti i record doppi mentre effettua l'ordinamento. Ad esempio, com'è possibile vedere solamente i valori distinti di Eta presenti nella tabella LAVORATORE? La query da eseguire è la seguente:

```
select DISTINCT Eta
  from LAVORATORE;
```

Come per la query precedente, questa query non ha condizioni di limitazione, per cui viene eseguita un'operazione di TABLE ACCESS FULL per recuperare i dati dalla tabella LAVORATORE. Tuttavia, la funzione DISTINCT indica all'ottimizzatore di restituire solo i valori distinti per la colonna Eta.

Per eliminare i valori di Eta doppi, l'ottimizzatore utilizza un'operazione SORT UNIQUE.

Per risolvere la query, l'ottimizzatore prende i record restituiti dall'operazione di TABLE ACCESS FULL e li ordina tramite un'operazione di SORT UNIQUE. Nessun record viene visualizzato all'utente fino a che tutti i record non sono stati elaborati. Nella Figura 36.3 viene mostrato l'ordine delle operazioni per la query dell'esempio comprendente un'operazione di SORT UNIQUE.

Oltre a essere utilizzata dalla funzione DISTINCT, l'operazione di SORT UNIQUE viene coinvolta quando si richiamano 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 di MERGE JOIN e mai da sola. Le implicazioni di SORT JOIN sulle prestazioni delle unioni sono descritte nel paragrafo "Operazioni di unione" di questo capitolo.

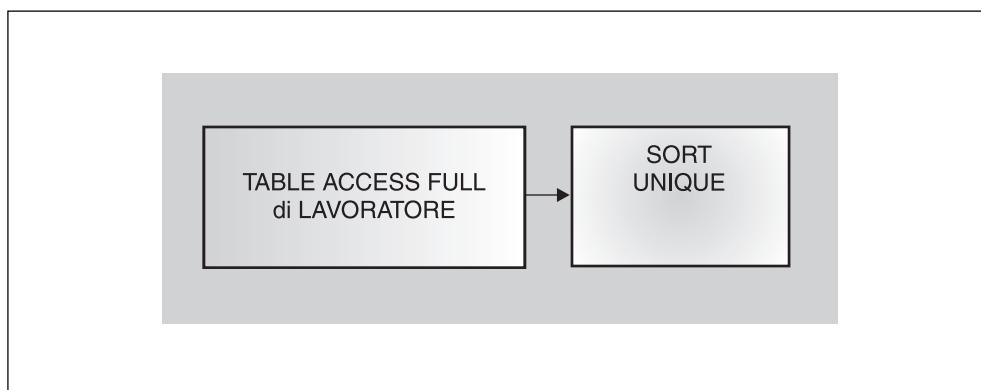


Figura 36.3 Ordine delle operazioni per SORT UNIQUE.

Raggruppamento di righe

Esistono due operazioni interne di ORACLE che consentono di ordinare le righe mentre raggruppano assieme i record uguali. Le due operazioni, SORT AGGREGATE e SORT GROUP BY, vengono utilizzate assieme alle funzioni di raggruppamento (quali MIN, MAX e COUNT). La sintassi della query determina quale operazione viene utilizzata.

Nella query seguente viene selezionato dalla tabella LAVORATORE il valore massimo di Eta:

```
select MAX(Eta)
      from LAVORATORE;
```

Per risolvere la query, l'ottimizzatore esegue due operazioni separate. Per prima un'operazione di TABLE ACCESS FULL per selezionare i valori di Eta dalla tabella, quindi l'analisi delle righe tramite un'operazione di SORT AGGREGATE. Questa operazione restituisce all'utente il valore massimo di Eta.

Se la colonna Eta è indicizzata, l'indice può essere utilizzato per risolvere query riguardanti il valore massimo o minimo (come descritto in uno dei paragrafi precedenti, "Operazioni che utilizzano indici"). Poiché la colonna Eta non è indicizzata, è richiesta un'operazione di ordinamento. Il valore massimo di Eta non viene restituito da questa query fino a che non sono stati letti tutti i record ed è stata completata l'operazione di SORT AGGREGATE.

L'operazione di SORT AGGREGATE è stata utilizzata nell'esempio precedente perché nella query non vi era 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 di lavoratori presenti in ciascun Alloggio? La query che segue seleziona il conteggio di ciascun valore di Alloggio dalla tabella LAVORATORE per mezzo della clausola group by:

```
select Alloggio, COUNT(*)
      from LAVORATORE
     group by Alloggio;
```

La query del listato precedente restituisce un record per ciascun valore distinto di Alloggio. Per ciascun valore di Alloggio, il numero delle sue ricorrenze nella tabella LAVORATORE viene calcolato e visualizzato nella colonna COUNT(*)).

Per risolvere questa query ORACLE esegue innanzitutto una scansione completa della tabella (non vi sono condizioni di limitazione). Poiché viene specificata una clausola group by, le righe restituite dall'operazione di TABLE ACCESS FULL vengono elaborate da un'operazione di SORT GROUP BY. I record vengono restituiti all'utente dopo che ciascuna riga è stata ordinata in gruppi e che è stato calcolato il conteggio di ciascun gruppo. 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 accedono 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 di utilizzare le operazioni di ordinamento e di raggruppamento che costringano gli utenti ad aspettare che i record vengano elaborati.

Se possibile, è opportuno scrivere query che consentano agli utenti dell'applicazione di ricevere velocemente i record nel momento in cui la query viene risolta.

Quanto meno operazioni di ordinamento e di raggruppamento vengono eseguite, tanto più velocemente viene restituito all'utente il primo record trovato. In una transazione batch le prestazioni della query vengono misurate in base al tempo complessivo impiegato per il completamento della query, 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.

Operazioni che utilizzano RowNum

Le query che utilizzano la pseudocolonna RowNum eseguono o l'operazione COUNT o l'operazione COUNT STOPKEY per incrementare il contatore RowNum. Se viene applicata una condizione di limitazione alla pseudocolonna RowNum, del tipo:

```
where RowNum <= 10
```

viene utilizzata l'operazione COUNT STOPKEY. Se non viene specificata 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.

La query seguente utilizza l'operazione COUNT, poiché fa riferimento alla pseudocolonna RowNum:

```
select Nome,
       RowNum
  from LAVORATORE;
```

Per risolvere la query precedente, l'ottimizzatore esegue una scansione completa della tabella (un'operazione TABLE ACCESS FULL nei confronti della tabella LAVORATORE), seguita da un'operazione COUNT per generare i valori di RowNum per ogni riga restituita. L'operazione COUNT non ha necessità di attendere che l'intero gruppo di record sia disponibile. Nel momento in cui un record viene restituito dalla tabella LAVORATORE, il contatore RowNum viene incrementato e viene determinato il RowNum per quel record.

Nell'esempio seguente viene imposta una condizione di limitazione sulla pseudocolonna RowNum:

```
select Nome,
       RowNum
  from LAVORATORE
 where RowNum < 10;
```

Per imporre la condizione di limitazione, l'ottimizzatore sostituisce l'operazione COUNT con un'operazione COUNT STOPKEY. Quest'ultima confronta il valore incrementato della pseudocolonna RowNum con la condizione di limitazione fornita. Quando il valore di RowNum supera il valore specificato nella condizione di limitazione, la query non restituisce più alcuna riga.

Ogni volta che viene utilizzata la pseudocolonna RowNum, l'ottimizzatore esegue un'operazione COUNT o COUNT STOPKEY per generare i valori di RowNum e applicare le condizioni di limitazione definite dalla query.

UNION, MINUS e INTERSECT

Le funzioni UNION, MINUS e INTERSECT consentono di elaborare e confrontare i risultati di più query. Ciascuna di queste funzioni ha un'operazione associata con lo stesso nome.

La query seguente seleziona tutti i valori della colonna Nome dalla tabella PROSPETTIVA assieme ai valori della colonna Nome della tabella DIPENDENTE:

```
select Nome
      from PROSPETTIVA
      UNION
select Nome
      from DIPENDENTE;
```

L'ottimizzatore esegue ciascuna query separatamente e ne combina i risultati. La prima query è:

```
select Nome
      from PROSPETTIVA
```

Non vi sono condizioni di limitazione nella query, perciò l'accesso alla tabella PROSPETTIVA viene effettuato tramite un'operazione di TABLE ACCESS FULL.

La seconda query è:

```
select Nome
      from DIPENDENTE;
```

Anche in questa query non vi sono condizioni di limitazione, perciò anche l'accesso alla tabella DIPENDENTE viene effettuato con un'operazione di TABLE ACCESS FULL.

Poiché si esegue una UNION dei risultati delle due query, i due gruppi di risultati vengono uniti tramite un'operazione di UNION-ALL. Con la funzione UNION si forza ORACLE a eliminare i record duplicati, quindi il gruppo di risultati viene elaborato da un'operazione di SORT UNIQUE prima che i record vengano restituiti all'utente. Nella Figura 36.4 viene mostrato l'ordine delle operazioni per quanto riguarda la funzione UNION.

Se la query ha utilizzato una funzione UNION ALL al posto di UNION, l'operazione di SORT UNIQUE (si veda la Figura 36.4) non è necessaria. In tal caso, la query risulta essere:

```
select Nome
      from PROSPETTIVA
      UNION ALL
select Nome
      from DIPENDENTE;
```

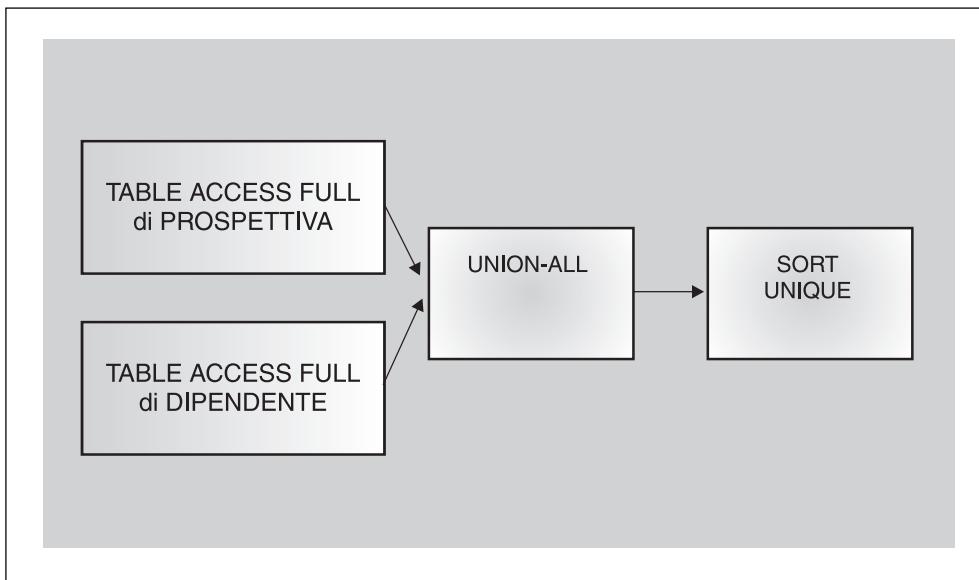


Figura 36.4 Ordine delle operazioni per la funzione UNION.

Quando viene elaborata quest'ultima query, l'ottimizzatore esegue le scansioni delle tabelle richieste dalle due query, seguite da un'operazione di UNION-ALL. Non è richiesta alcuna operazione di SORT UNIQUE, poiché la funzione UNION ALL non elimina i record duplicati.

Nel elaborare la query UNION, l'ottimizzatore indirizza separatamente ciascuna delle query destinate alla UNION. Nonostante in tutti gli esempi mostrati nei listati precedenti siano state coinvolte query semplici con scansioni di tabelle complete, le query destinate a UNION possono essere molto complesse, con corrispondenti percorsi di esecuzione complessi. I risultati non vengono restituiti all'utente fino a che tutti i record non sono stati elaborati.

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 Nome provenienti dalle tabelle LAVORATORE e PROSPETTIVA. Un valore di Nome viene restituito dalla query solo se esiste nella tabella LAVORATORE ma non in PROSPETTIVA.

```

select Nome
  from LAVORATORE
MINUS
select Nome
  from PROSPETTIVA;
  
```

Quando questa query viene eseguita, le due query destinate a MINUS vengono elaborate separatamente. La prima query:

```

select Nome
  from LAVORATORE
  
```

richiede una scansione completa della tabella LAVORATORE.

La seconda query:

```
select Nome
  from PROSPETTIVA;
```

richiede una scansione completa della tabella PROSPETTIVA.

Per eseguire la funzione MINUS, ciascun gruppo di record restituito dalle scansioni complete delle due tabelle viene ordinato attraverso un'operazione di SORT UNIQUE (con la quale vengono ordinate le righe ed eliminate quelle duplicate). 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 36.5.

Come viene 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 di SORT UNIQUE non sono state completate. Come per l'esempio della query UNION, la query mostrata nell'esempio riguardante l'operazione MINUS ha 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 delle due query per stabilire le righe che hanno in comune. In questo esempio viene mostrata una query semplice, che può essere utilizzata come base per una più complessa.

La query che segue determina i valori di Nome che si trovano sia nella tabella PROSPETTIVA sia nella tabella DIPENDENTE:

```
select Nome from PROSPETTIVA
INTERSECT
select Nome from DIPENDENTE;
```

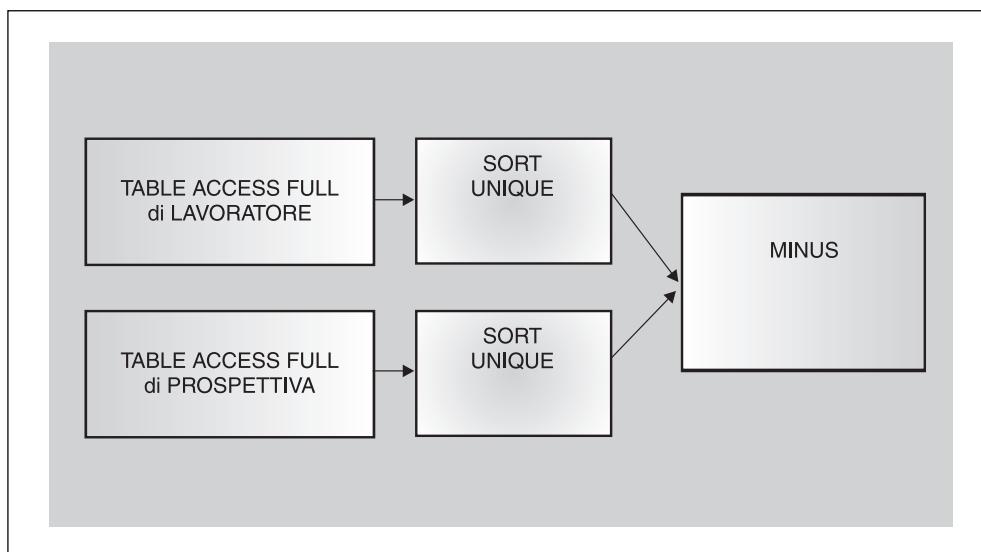


Figura 36.5 Ordine delle operazioni per la funzione MINUS.

Per elaborare la query `INTERSECT`, l'ottimizzatore inizia con il valutare ciascuna query separatamente. La prima query:

```
select Nome from PROSPETTIVA
```

richiede un'operazione di TABLE ACCESS FULL della tabella PROSPETTIVA. La seconda query,

```
select Nome from DIPENDENTE;
```

richiede un'operazione di TABLE ACCESS FULL nei confronti della tabella DIPENDENTE. I risultati delle scansioni delle due tabelle vengono elaborati separatamente tramite operazioni di SORT UNIQUE. In altre parole, vengono ordinate le righe provenienti dalla tabella PROSPETTIVA e quelle della tabella DIPENDENTE. I risultati dei due ordinamenti vengono confrontati tramite l'operazione di INTERSECTION, la quale restituisce i valori di Nome, a loro volta restituiti da entrambi gli ordinamenti.

Nella Figura 36.6 viene mostrato l'ordine delle operazioni per quanto riguarda la funzione `INTERSECT` dell'esempio precedente.

Come si può vedere dalla Figura 36.6, il percorso esecutivo di una query che utilizza una funzione `INTERSECT` richiede l'intervento di operazioni di SORT UNIQUE. Poiché le operazioni di 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 effettuare l'intersezione.

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.

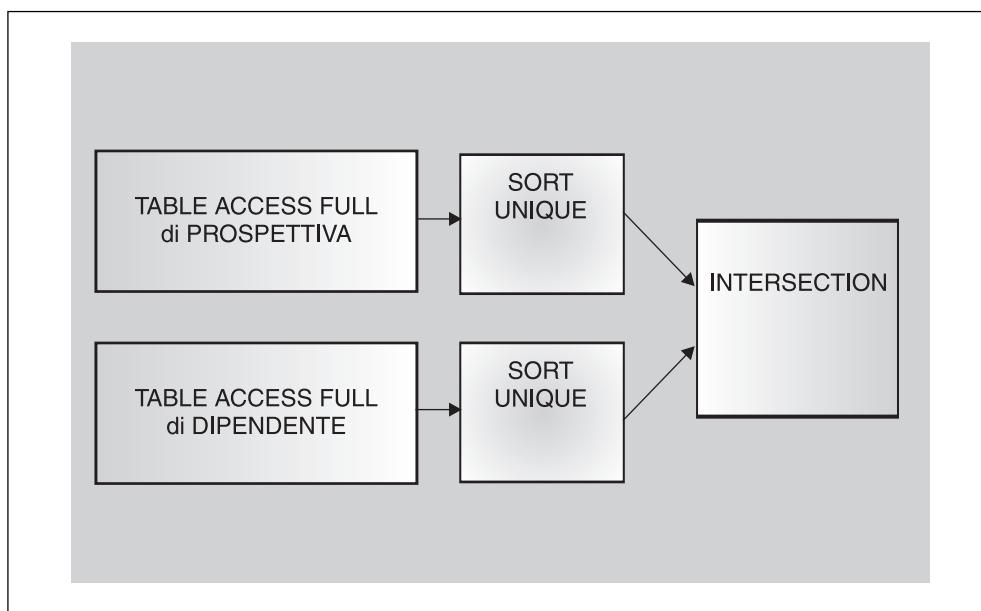


Figura 36.6 Ordine delle operazioni per la funzione `INTERSECT`.

Nell'esempio che segue viene illustrato in che modo i concetti e le operazioni trattati nei paragrafi precedenti possano essere messi in pratica per assemblare velocemente il processo esecutivo di una query. Nella query seguente viene analizzata la tabella LAVORATORE alla ricerca di valori della colonna Nome presenti sia nella tabella PROSPECTIVA sia nella tabella DIPENDENTE:

```
select Nome
  from LAVORATORE
 where Nome in
   (select Nome from PROSPECTIVA
    INTERSECT
    select Nome from DIPENDENTE);
```

Nella query precedente viene utilizzata come sottoquery la query INTERSECT dell'ultimo esempio. Se la colonna Nome della tabella LAVORATORE non è indicizzata, la prima operazione sarà la valutazione della sottoquery INTERSECT. Le righe restituite dalla sottoquery vengono utilizzate per generare un elenco di valori per la clausola in della query. Dopo che è stata eseguita la funzione INTERSECT, risultano noti i valori restituiti dalla sottoquery, perciò ora la query sarà del tipo:

```
select Nome
  from LAVORATORE
 where Nome in
   ('ADAH TALBOT', 'ELBERT TALBOT', 'PAT LAVAY',
    'WILFRED LOWELL');
```

Se la colonna Nome della tabella LAVORATORE non è indicizzata, questa query viene eseguita tramite un'operazione di TABLE ACCESS FULL sulla tabella LAVORATORE.

Si tenga presente che l'operazione di TABLE ACCESS FULL elimina dei record in base al criterio specificato nella clausola in, e i valori della clausola in non sono noti fino a che l'operazione di INTERSECTION non viene completata. Quindi l'operazione di TABLE ACCESS FULL dipende dal completamento dell'operazione di INTERSECTION, così come l'operazione di INTERSECTION dipende dal completamento delle operazioni di SORT UNIQUE.

Nonostante quello presentato sia un esempio complesso, la comprensione del modo in cui ciascuna componente viene elaborato consente di capire rapidamente il processo di esecuzione della query completa. Nella Figura 36.7 viene mostrato l'ordine delle operazioni per la query complessa.

Le funzioni UNION, MINUS e INTERSECT coinvolgono ciascuna l'elaborazione di gruppi di righe prima di restituire alcun risultato 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 influenza la velocità con cui viene restituita la prima riga all'utente.

Selezione di righe per l'aggiornamento

È possibile bloccare delle righe utilizzando la sintassi select for update. Ad esempio, la query seguente seleziona le righe dalla tabella LAVORATORE e le blocca per

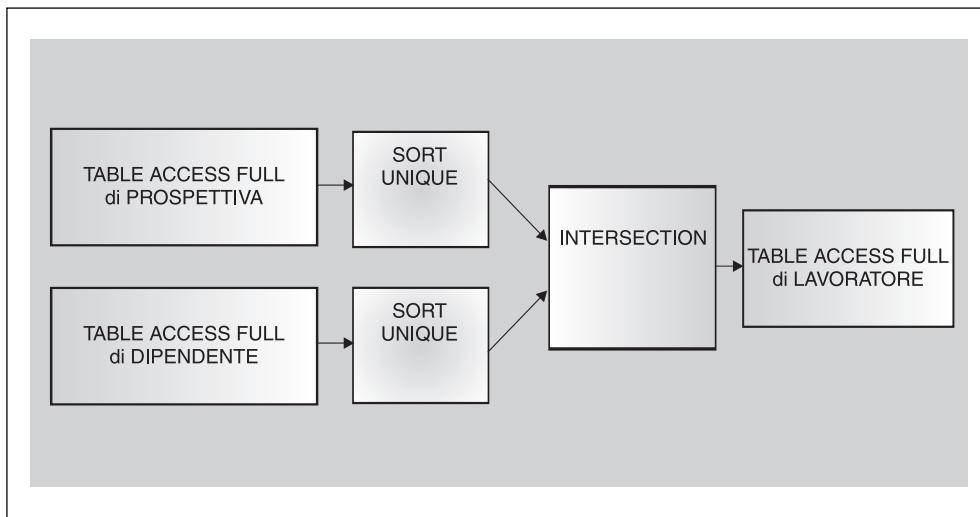


Figura 36.7 Ordine delle operazioni per l'esempio complesso di INTERSECT.

evitare che altri utenti se ne impadroniscano. L'utilizzo 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, perciò è necessario rieseguire il select for update dopo ciascun commit.

```
select *
  from LAVORATORE
 for update of Nome;
```

Quando viene eseguita la query precedente, l'ottimizzatore esegue innanzitutto un'operazione di TABLE ACCESS FULL per recuperare le righe dalla tabella LAVORATORE. L'operazione di TABLE ACCESS FULL restituisce le righe appena le trova; essa non attende che l'intera sequenza di dati da recuperare sia completa. Tuttavia, questa query deve eseguire una seconda operazione. L'operazione FOR UPDATE viene chiamata per bloccare i record. L'operazione FOR UPDATE è un'operazione basata su insiemi (analogamente alle operazioni di ordinamento), perciò non restituisce alcuna riga all'utente fino a che non è stato bloccato il gruppo di righe completo.

Selezione dalle viste

Quando si crea una vista, ORACLE registra la query su cui la vista si basa. Ad esempio, la vista seguente si basa sulla tabella LAVORATORE:

```
create or replace view OVER_20_LAVORATORE as
select Nome, Eta
  from LAVORATORE
 where Eta >20;
```

Quando si opera una selezione dalla vista OVER_20_LAVORATORE, l'ottimizzatore preleva i criteri dalla query dell'utente e li combina con il testo della query della vista. Ad esempio, se si esegue la query:

```
select *
  from OVER_20_LAVORATORE;
```

l'ottimizzatore da avvio alla query su cui si basa la vista OVER_20_LAVORATORE, proprio come se fosse stata inoltrata la query seguente:

```
select Nome, Eta
  from LAVORATORE
 where Eta >20;
```

Se nella query della vista si specificano delle condizioni di limitazione, esse, se possibile, vengono applicate al testo della query della vista. Ad esempio, se si esegue la query:

```
select Nome, Eta
  from OVER_20_LAVORATORE
 where Nome > 'M';
```

l'ottimizzatore combina le condizioni di limitazione dell'utente:

```
whereName > 'M'
```

con il testo della query della vista ed esegue la query:

```
select Nome, Eta
  from LAVORATORE
 where Eta >20
   and Nome > 'M';
```

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 dell'utente, le opzioni disponibili all'ottimizzatore aumentano. Ad esempio, se esiste un indice sulla colonna Nome della tabella LAVORATORE, la query combinata potrebbe utilizzare quell'indice (poiché Nome viene posta uguale a una sequenza di valori specificati nella clausola where).

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, viene risolta la vista stessa prima che il resto delle condizioni venga applicato. Si consideri la vista seguente:

```
create or replace view LAVORATORE_ETA_CONTA as
select Eta, COUNT(*) Conta_Eta
  from LAVORATORE
 group by Eta;
```

La vista LAVORATORE_ETA_CONTA visualizza una riga per ciascun valore di Eta distinto nella tabella LAVORATORE assieme al numero di record che possiede quel valore. La colonna Conta_Eta della vista LAVORATORE_ETA_CONTA registra il conteggio per ciascun valore distinto di Eta.

In che modo l'ottimizzatore elabora la query seguente della vista LAVORATORE_ETA_CONTA?

```
select *
  from LAVORATORE_ETA_CONTA
 where Conta_Eta > 1;
```

La query si riferisce alla colonna Conta_Eta della vista. Tuttavia, la clausola where della query non può essere combinata con il testo della query della vista, poiché Conta_Eta viene creata tramite un'operazione di raggruppamento. La clausola where non può essere applicata se non dopo che è stata completamente risolta l'interrogazione della vista LAVORATORE_ETA_CONTA.

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 scelte disponibili all'ottimizzatore e non restituiscono i record fino a che tutte le righe sono state elaborate; queste viste hanno scarse prestazioni se interrogate da utenti in linea.

Quando la query viene elaborata, per prima cosa l'ottimizzatore risolve la vista.

Poiché la query della vista è:

```
select Eta, COUNT(*) Conta_Eta
      from LAVORATORE
     group by Eta;
```

L'ottimizzatore legge i dati dalla tabella LAVORATORE tramite un'operazione di TABLE ACCESS FULL. Poiché viene utilizzata una clausola group by, le righe restituite dall'operazione di TABLE ACCESS FULL vengono elaborate tramite un'operazione di SORT GROUP BY. Due nuove operazioni, FILTER e NEW, elaborano quindi i dati. L'operazione FILTER viene utilizzata per eliminare le righe che si basano sul criterio specificato nella query:

```
where Conta_Eta > 10
```

L'operazione VIEW preleva l'output dell'operazione FILTER e restituisce l'output all'utente. Il flusso delle operazioni per la query della vista LAVORATORE_ETA_CONTA è mostrato nella Figura 36.8.

Se si utilizzano viste che possiedono clausole group by, le righe non vengono restituite fino a che ciascuna di esse non è stata elaborata dalla vista stessa. Di conseguenza può trascorrere parecchio tempo fino a che la prima riga venga restituita dalla query 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 proprie viste, si incrementerebbero le probabilità che il testo della vista possa essere unito con quello della query che la richiama la vista e di conseguenza le prestazioni potrebbero migliorare (benché la query possa utilizzare altri gruppi di operazioni il cui impatto si riveli negativo sulle prestazioni).

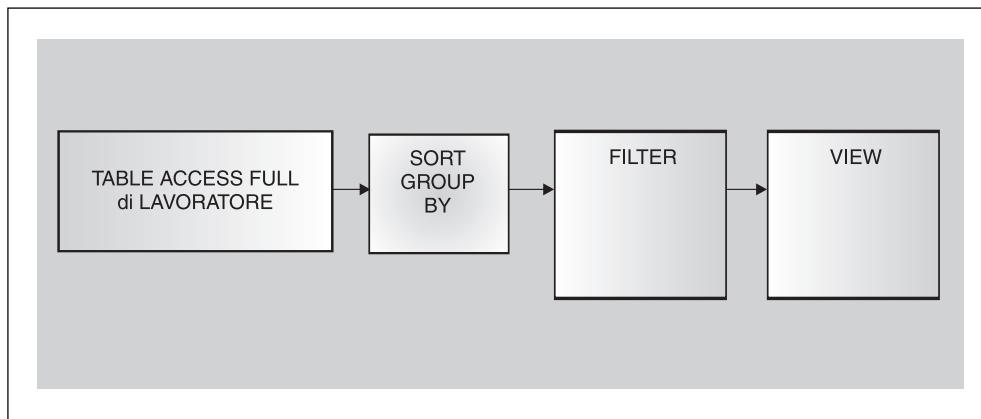


Figura 36.8 Ordine delle operazioni per la query della vista.

Selezioni da sottoquery

Ogni volta che è possibile, l'ottimizzatore combina il testo proveniente da una sottoquery con il resto della query. Ad esempio, si consideri la query seguente:

```

select *
  from LAVORATORE
 where Alloggio =
       (select Alloggio
        from ALLOGGIO
       where Direttore = 'KEN MULLER'
  
```

L'ottimizzatore, nel valutare la query precedente, determina che la query è funzionalmente equivalente alla seguente unione delle tabelle LAVORATORE e ALLOGGIO:

```

select LAVORATORE.*
  from LAVORATORE, ALLOGGIO
 where LAVORATORE.Alloggio = ALLOGGIO.Alloggio
   and Direttore = 'KEN MULLER';
  
```

Se la query non viene scritta come un'unione, l'ottimizzatore dispone di alcune operazioni per elaborare i dati (come descritto nel paragrafo “Operazioni che eseguono unioni” più avanti in questo capitolo).

Se la query non può essere risolta come un'unione, allora viene risolta prima che la parte restante del testo venga elaborato rispetto ai dati; si tratta di un meccanismo simile al modo in cui l'operazione VIEW viene utilizzata dalle viste. In realtà l'operazione VIEW viene utilizzata per le sottoquery se queste ultime non possono essere unite con il resto della query.

La query seguente seleziona dalla tabella LAVORATORE i lavoratori con età superiore alla media (in base alla colonna Eta):

```

select Nome
  from LAVORATORE
  
```

```
where Eta >
      (select AVG(Eta)
       from LAVORATORE);
```

Nell'elaborare la query precedente, l'ottimizzatore non può unire la sottoquery e il testo della query che la richiama. Di conseguenza, la sottoquery viene elaborata per prima; si ha un'operazione di TABLE ACCESS FULL della tabella LAVORATORE, seguita da un'operazione di SORT AGGREGATE per stabilire il valore medio di Eta. Una volta stabilito, questo valore viene restituito alla query principale, fornendo così il valore con il quale confrontare gli altri valori di Eta (tramite un'operazione FILTER).

Le sottoquery che dipendono da operazioni di raggruppamento presentano gli stessi problemi di regolazione delle viste che contengono operazioni di raggruppamento. Le righe recuperate da queste sottoquery devono essere completamente elaborate prima di applicare le restanti condizioni di limitazione.

Quando il testo della query viene unito con il testo della vista, aumentano le opzioni disponibili per l'ottimizzatore. Ad esempio, la combinazione delle condizioni di limitazione della query con le condizioni di limitazione della vista può consentire l'utilizzo di un indice prima inutilizzabile durante l'esecuzione della query.

L'unione automatica del testo della query e di quello della vista può essere disabilitata tramite gli hint, come descritto nel paragrafo seguente.

Hint e parametri

Le applicazioni che sono state precedentemente regolate per funzionare con un testo della vista non unito a quello della query possono disabilitare le operazioni di unione eseguite dall'ottimizzatore. Per consentire alle viste e alle query di essere elaborate senza l'unione del rispettivo testo, è necessario modificare il parametro _optimizer_undo_changes del file init.ora. Per disabilitare l'unione delle sottoquery e del testo della query chiamante, si imposti questo parametro al valore TRUE all'interno del proprio file init.ora, come viene mostrato nel listato che segue:

```
_optimizer_undo_change = TRUE
```

Ulteriori aspetti di messa a punto

Se si sta effettuando la messa a punto di query destinate a essere utilizzate da 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.

Ad esempio, l'UNION di query in cui ciascuna query contiene una clausola group by, richiede degli ordinamenti annidati; viene richiesta un'operazione di ordinamento per ciascuna delle query, seguita da un'operazione di 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 stanno utilizzando 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. Ad esempio, è possibile interrogare i dati da due sorgenti separate e riportare i risultati attraverso un'unica query utilizzando la funzione UNION. Se non è possibile che le due query restituiscano le stesse righe, si può rimpiazzare la funzione UNION con UNION ALL ed evitare così l'operazione SORT UNIQUE eseguita dalla funzione UNION.

36.5 Operazioni di unione

Spesso un'unica query ha la necessità di selezionare colonne da più tabelle. A questo scopo le tabelle vengono unite nell'istruzione SQL; le tabelle vengono elencate nella clausola from e le condizioni di unione vengono elencate nella clausola where. Nell'esempio che segue vengono unite le tabelle LAVORATORE e ALLOGGIO, in base ai valori comuni della colonna Alloggio

```
select LAVORATORE.Nome, ALLOGGIO.Direttore
  from LAVORATORE, ALLOGGIO
 where LAVORATORE.Alloggio = ALLOGGIO.Alloggio
```

Le condizioni di unione possono agire come condizioni di limitazione per l'unione stessa. Poiché la colonna LAVORATORE.Alloggio è posta uguale al valore nella clausola where, l'ottimizzatore può, durante l'esecuzione della query, utilizzare un indice sulla colonna LAVORATORE.Alloggio.

Se la colonna ALLOGGIO.Alloggio dispone di un indice, anche questo indice può essere preso in considerazione dall'ottimizzatore.

ORACLE mette in atto tre modalità per elaborare le unioni: le operazioni di MERGE JOIN, le operazioni NESTED LOOPS (ORACLE7.3) e le operazioni di HASH JOIN. In base alle condizioni presenti nella propria query, agli indici disponibili e (per il CBO) alle informazioni statistiche disponibili, l'ottimizzatore sceglie quale operazione di unione utilizzare. A seconda della natura della propria applicazione e delle query, si può decidere di forzare l'ottimizzatore a utilizzare un metodo differente dalla sua prima scelta. Nei paragrafi che seguono vengono analizzate le caratteristiche dei diversi metodi di unione e le condizioni in cui ciascuno di essi si rivela più utile.

Modalità di ORACLE nella gestione dell'unione di più di due tabelle

Se in una query vengono unite più di due tabelle fra loro, l'ottimizzatore tratta la query come un insieme di più unioni. Ad esempio, se la query esegue l'unione di tre tabelle, l'ottimizzatore unisce prima due tabelle assieme e successivamente i risultati ottenuti con la terza tabella. La dimensione dei risultati dell'unione iniziale influenza sulle prestazioni delle restanti unioni.

Se la dimensione dei risultati ottenuti con la prima unione è di una certa entità, la seconda unione dovrà elaborare molte righe.

Ad esempio, se nella query vengono unite tra tabelle di dimensione differente, quali una piccola tabella di nome BREVE, 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 l'unione di MEDIA con GRANDE restituisce molte righe, l'unione del risultato ottenuto con la tabella BREVE può richiedere parecchio lavoro. In alternativa, se vengono per prime unite le tabelle BREVE e MEDIA, l'unione del risultato e della tabella GRANDE può ridurre la quantità di lavoro eseguito dalla query. Nel paragrafo relativo alle operazioni di unione, oltre che nel paragrafo che spiega come visualizzare i percorsi esecutivi e il comando set autotrace on, viene spiegato come interpretare l'ordine delle unioni.

MERGE JOIN

Concettualmente, l'operazione MERGE JOIN è la più semplice da capire. In questa operazione, i due input dell'unione vengono elaborati separatamente, ordinati e uniti. Le operazioni di MERGE JOIN vengono comunemente utilizzate quando non vi sono indici disponibili per le condizioni di limitazione della query.

Nella query seguente vengono unite le tabelle LAVORATORE e ALLOGGIO. Se nessuna delle tabelle possiede un indice sulla propria colonna Alloggio, non vi sono indici che possano essere utilizzati durante la query (poiché esistono altre condizioni di limitazione nella query).

```
select LAVORATORE.Nome, ALLOGGIO.Direttore
  from LAVORATORE, ALLOGGIO
 where LAVORATORE.Alloggio = ALLOGGIO.Alloggio;
```

Per risolvere la query l'ottimizzatore può scegliere di eseguire un'operazione di MERGE JOIN delle tabelle. Questa operazione richiede che ciascuna delle tabelle venga letta separatamente tramite un'operazione di TABLE ACCESS FULL. L'insieme di righe restituite dalla lettura della tabella LAVORATORE viene ordinato tramite un'operazione di SORT JOIN, così come anche l'insieme di righe restituite dalla lettura della tabella ALLOGGIO viene ordinato da un'operazione di SORT JOIN separata. Nella Figura 36.9 viene mostrato l'ordine delle operazioni relativo all'operazione dell'esempio.

Quando si utilizza un'operazione di MERGE JOIN per unire due gruppi di record, ciascun gruppo viene elaborato separatamente prima di essere unito. L'operazione di MERGE JOIN non può iniziare fino a che non ha ricevuto i dati da entrambe le operazioni di SORT JOIN che le forniscono l'input. Le operazioni di SORT JOIN, a loro volta, non forniscono dati al MERGE JOIN fino a che tutte le righe non sono state ordinate.

Poiché l'operazione di MERGE JOIN deve attendere il completamento di due operazioni separate di SORT JOIN, un'unione che utilizza MERGE JOIN ha normalmente prestazioni scarse per gli utenti in linea. Questo problema è dovuto al ritardo nel restituire agli utenti la prima riga dell'unione. Se la tabella aumenta 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 ha un impatto negativo sulle prestazioni complessive della query.

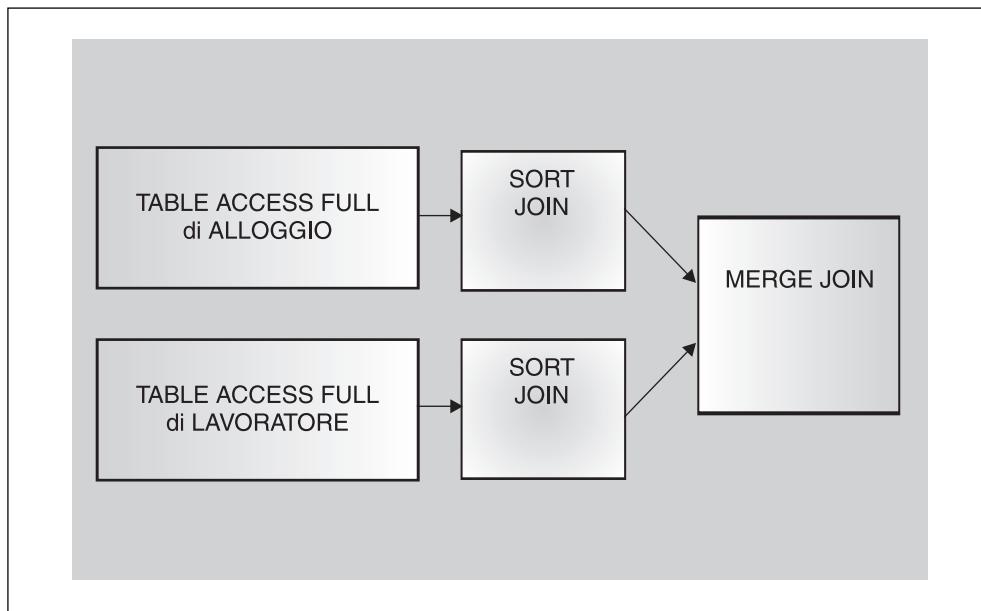


Figura 36.9 Ordine delle operazioni per MERGE JOIN.

Si consideri nuovamente l'unione delle tabelle LAVORATORE e ALLOGGIO. Che cosa accade se la tabella ALLOGGIO possiede 100 voci e la tabella LAVORATORE 10000? Le operazioni di TABLE ACCESS FULL e SORT JOIN per la tabella ALLOGGIO vengono completate velocemente. Tuttavia, l'operazione di MERGE JOIN non può iniziare fino a che non sono state completate le operazioni di TABLE ACCESS FULL e SORT JOIN della tabella LAVORATORE, che richiedono più tempo.

Le operazioni di MERGE JOIN sono più efficienti se le tabelle sono di ugual dimensione. Poiché le operazioni di MERGE JOIN richiedono una scansione e un ordinamento completi delle tabelle coinvolte, dovrebbero essere effettuate solo se entrambe le tabelle sono molto piccole o molto grandi. Se entrambe le tabelle sono molto piccole, il processo di lettura e ordinamento viene completato velocemente. Se entrambe sono molto grandi, le operazioni di ordinamento e di scansione richieste dalle operazioni di MERGE JOIN possono avvantaggiarsi delle opzioni parallele di ORACLE.

ORACLE è in grado di eseguire operazioni in parallelo, in quanto può far partecipare più processori all'esecuzione di un unico comando. Tra le operazioni che possono essere eseguite in parallelo vi sono quelle di TABLE ACCESS FULL e le operazioni di ordinamento. Poiché MERGE JOIN le utilizza entrambe, può ottenere un avvantaggiarsi delle opzioni parallele di ORACLE. L'esecuzione in parallelo di query che coinvolgono operazioni di MERGE JOIN spesso incrementa le prestazioni (purché vi siano adeguate risorse di sistema disponibili per il supporto delle operazioni parallele).

Se si utilizza un'operazione di MERGE JOIN, viene normalmente indicato che non vi sono indici disponibili per le condizioni di unione. Se vi fosse un indice disponibile, l'optimizer lo sceglie.

nibile per tali condizioni, ORACLE potrebbe eseguire un'unione di tipo NESTED LOOPS. A partire da ORACLE7.3 l'ottimizzatore può invece scegliere dinamicamente di eseguire un'operazione di HASH JOIN. Le operazioni di HASH JOIN e le unioni NESTED LOOPS vengono descritte nei paragrafi seguenti, quindi sono riportati gli hint che consentono all'utente di influenzare il metodo di unione selezionato dall'ottimizzatore.

NESTED LOOPS

Prima dell'introduzione delle unioni hash in ORACLE7.3, le operazioni di NESTED LOOPS rappresentavano il principale metodo di unione per le query eseguite dagli utenti in linea. Le operazioni di NESTED LOOPS consentono di unire due tabelle tramite un metodo di risoluzione ciclica: i record vengono recuperati da una tabella e, per ciascun record recuperato, viene effettuato un accesso alla seconda tabella. L'accesso alla seconda tabella avviene in base a un indice.

Nel listato che segue viene riproposta la query del paragrafo precedente, con MERGE JOIN:

```
select LAVORATORE.Nome, ALLOGGIO.Direttore
      from LAVORATORE, ALLOGGIO
     where LAVORATORE.Alloggio = ALLOGGIO.Alloggio;
```

Per effettuare un'unione NESTED LOOPS deve essere disponibile un indice da utilizzare con la query. Nell'esempio di MERGE JOIN si supponeva che le due tabelle non avessero indici. Nel listato che segue viene creata una chiave primaria sulla tabella ALLOGGIO. Poiché la limitazione PRIMARY KEY crea implicitamente un indice unico sulla colonna Alloggio della tabella ALLOGGIO, risulta disponibile un indice per l'unione.

```
alter table ALLOGGIO add
constraint ALLOGGIO_PK primary key (Alloggio);
```

Quando viene creata la limitazione ALLOGGIO_PK, viene automaticamente creato un indice unico di nome ALLOGGIO_PK sulla colonna Alloggio.

Poiché la colonna Alloggio della tabella omonima viene utilizzata come parte della condizione di unione nella query, per risolvere la query può essere utilizzato l'indice ALLOGGIO_PK. Quando si esegue la query, per effettuare l'unione può essere utilizzata un'operazione di NESTED LOOPS.

Per effettuare un'unione NESTED LOOPS, l'ottimizzatore deve innanzitutto selezionare la *tabella guida*, ovvero la tabella che viene letta per prima (solitamente attraverso un'operazione di TABLE ACCESS FULL). Per ciascun record nella tabella guida viene interrogata la seconda tabella all'interno dell'unione. La query di esempio unisce le tabelle LAVORATORE e ALLOGGIO in base ai valori della colonna Alloggio. Poiché sulla colonna Alloggio della tabella omonima è disponibile un indice e nessun indice simile è invece disponibile per la tabella LAVORATORE, quest'ultima viene utilizzata come tabella guida della query.

Durante l'esecuzione di NESTED LOOPS, un'operazione di TABLE ACCESS FULL seleziona tutti i record dalla tabella LAVORATORE. Viene sondato l'indice ALLOGGIO_PK della tabella ALLOGGIO per sapere se contiene una voce per il

valore corrispondente al record corrente prelevato dalla tabella LAVORATORE. Se esiste una corrispondenza, viene prelevato il RowID per la riga di ALLOGGIO corrispondente e la riga in questione viene selezionata dalla tabella ALLOGGIO tramite un'operazione di TABLE ACCESS BY ROWID. Nella Figura 36.10 è mostrato il flusso delle operazioni per l'unione NESTED LOOPS.

Come si può vedere dalla Figura 36.10, in un'unione NESTED LOOPS vengono coinvolte tre operazioni di accesso ai dati: TABLE ACCESS FULL, INDEX UNIQUE SCAN e TABLE ACCESS FULL BY ROWID. Ciascuno di questi metodi di accesso ai dati restituisce i record alle operazioni successive, appena questi vengono trovati, senza attendere di recuperare l'intero insieme di record. Poiché queste operazioni sono in grado di fornire velocemente agli utenti la prima riga che corrisponde ai criteri specificati, le unioni NESTED LOOPS vengono riservate normalmente per gli utenti in linea.

Nell'implementazione di unioni NESTED LOOPS è necessario considerare la dimensione delle tabella guida. Se la tabella guida è grande, l'operazione di TABLE ACCESS FULL eseguita su di essa può influire negativamente sulle prestazioni della query. Se sono disponibili più indici, ORACLE seleziona una tabella guida per la query. Il metodo della selezione per tabelle guida dipende dall'ottimizzatore che si sta utilizzando. Se si sta utilizzando 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 si sta utilizzando l'RBO e sono disponibili degli indici per ciascuna delle condizioni di unione, la tabella guida risulta solitamente essere quella tabella elencata per ultima nella clausola from.

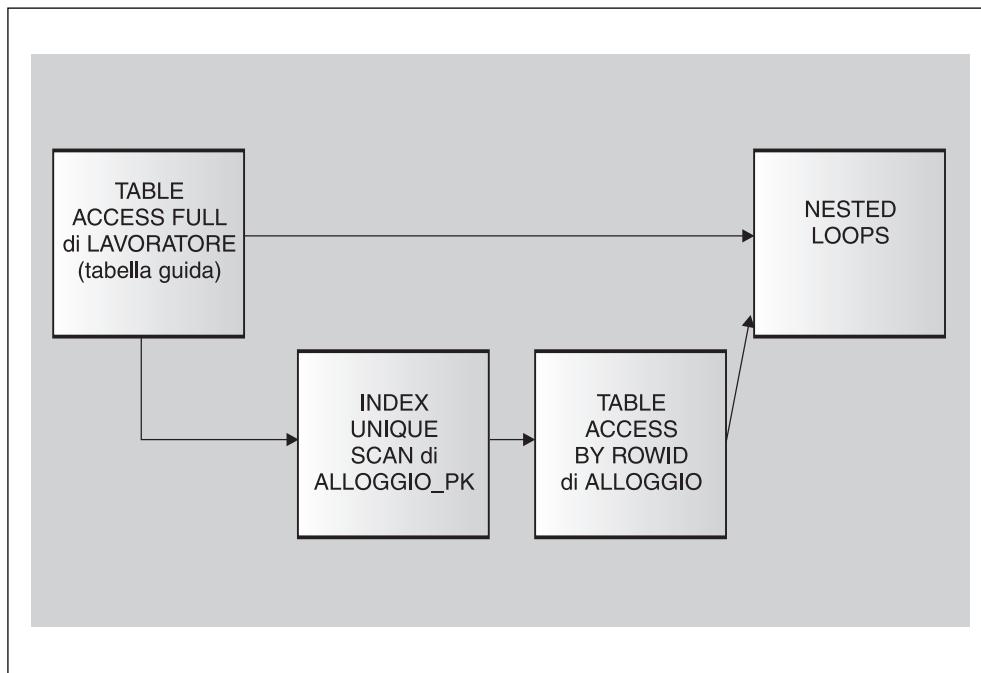


Figura 36.10 Ordine delle operazioni per NESTED LOOPS.

Quando si uniscono tre tavole, ORACLE esegue due unioni separate: un'unione di due tavole che genera un insieme di record e successivamente un'unione tra questo insieme di record e la terza tabella. Se si utilizzano unioni NESTED LOOPS, risulta critico l'ordine in cui le tavole vengono unite. L'output proveniente dalla prima unione genera un insieme di record e quest'ultimo viene utilizzato come tabella guida per la seconda unione. Nella Figura 36.11 è presentata la sequenza delle operazioni per un'unione NESTED LOOPS di tre tavole: LAVORATORE, ALLOGGIO e una terza tabella.

Confrontando la Figura 36.11 con la Figura 36.10 si può vedere come, in questo esempio, l'unione di due tavole formi la base per l'unione di tre tavole. Come viene mostrato nella Figura 36.11, la dimensione dell'insieme di record restituito dalla prima unione influisce sulle prestazioni della seconda unione e in tal modo può avere un impatto significativo sulle prestazioni della query complessiva. È consigliabile cercare di unire per prime le tavole più piccole, più selettive, per fare in modo che l'impatto di queste unioni sulle successive sia trascurabile. Se nella prima unione di una query vengono assemblate tavole di grandi dimensioni, la dimensione di queste tavole influirà su ciascuna unione successiva e avrà un impatto negativo sulle prestazioni complessive della query.

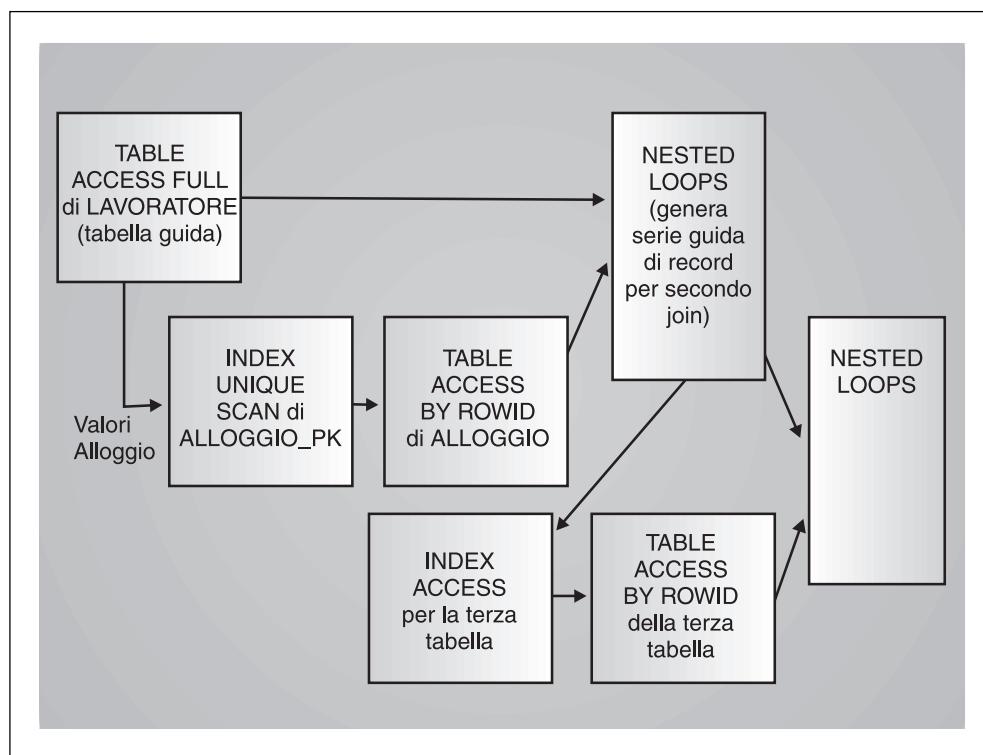


Figura 36.11 Ordine delle operazioni per un'unione NESTED LOOP a tre tavole.

Le unioni NESTED LOOPS risultano utili quando le tabelle coinvolte nell'unione sono di uguali dimensioni, infatti si può utilizzare la tabella più piccola come tabella guida ed effettuare la selezione 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

A partire da ORACLE7.3 l'ottimizzatore può scegliere in modo dinamico di effettuare unioni utilizzando l'operazione HASH JOIN. Questa operazione confronta due tabelle in memoria. Durante un'unione di questo tipo, la prima tabella viene analizzata tramite un'operazione di TABLE ACCESS FULL e il database applica le funzioni di hash ai dati per preparare la tabella all'unione. I valori provenienti dalla seconda tabella vengono quindi letti (anch'essi tramite un'operazione di TABLE ACCESS FULL) e viene utilizzata la funzione di hash per confrontare la seconda tabella con la prima. Le righe trovate corrispondenti ai criteri ricercati vengono restituite all'utente.

NOTA *Nonostante possiedano nomi simili, le unioni di hash non hanno nulla a che vedere con i cluster di hash o con le operazioni di TABLE ACCESS HASH trattante più avanti in questo capitolo.*

L'ottimizzatore può scegliere di eseguire unioni di hash anche se gli indici sono disponibili. Nella query d'esempio mostrata nel listato seguente, le tabelle LAVORATORE e ALLOGGIO vengono unite sulla colonna Alloggio:

```
select LAVORATORE.Nome, ALLOGGIO.Direttore
      from LAVORATORE, ALLOGGIO
     where LAVORATORE.Alloggio = ALLOGGIO.Alloggio;
```

La tabella ALLOGGIO ha un indice unico sulla sua colonna Alloggio. Poiché l'indice è disponibile e può essere utilizzato per valutare le condizioni di unione, l'ottimizzatore può scegliere di eseguire un'unione NESTED LOOPS delle due tabelle. Tuttavia, l'ottimizzatore potrebbe anche scegliere di eseguire un'unione di hash. Se viene eseguita un'unione di questo tipo, ciascuna tabella viene letta tramite un'operazione di TABLE ACCESS FULL separata. I dati ricavati dalle scansioni della tabella fungono da input per un'operazione HASH JOIN.

Poiché le unioni di hash si basano su scansioni complete delle tabelle, risultano più efficaci quando si utilizzano le opzioni parallele di ORACLE per migliorare le prestazioni delle letture complete. Questo tipo di unione utilizza la memoria (la quantità di memoria allocata per un'operazione di questo tipo è specificata attraverso i parametri di init.ora), perciò le applicazioni che utilizzano in modo intensivo le unioni di hash possono avere la necessità di incrementare la quantità di memoria disponibile nella SGA del database.

Nella Figura 36.12 viene mostrato l'ordine delle operazioni per un'unione di hash. Come si può vedere dalla figura, l'unione di hash non si basa su operazioni che elaborano gruppi di righe.

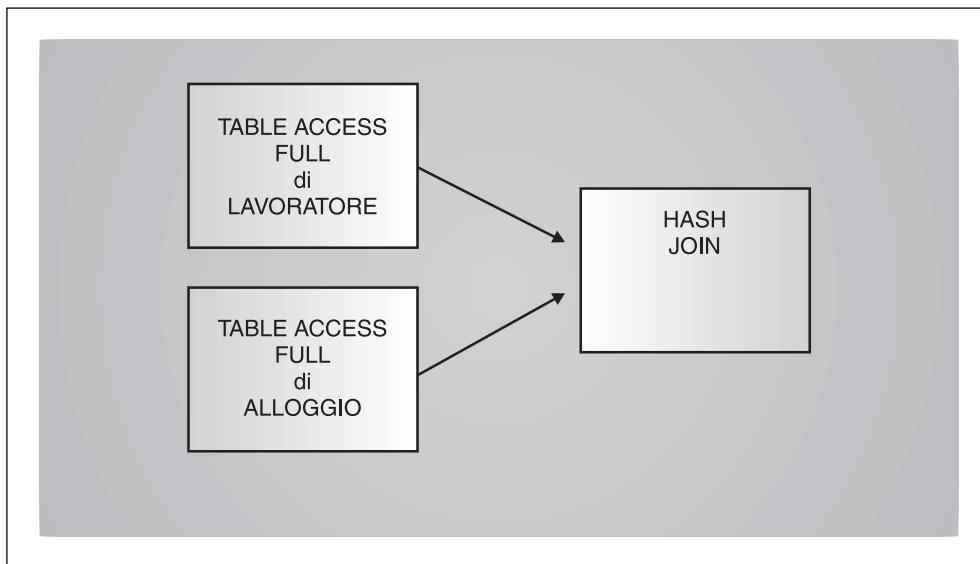


Figura 36.12 Ordine delle operazioni per HASH JOIN.

Le operazioni coinvolte nelle unioni di hash restituiscono velocemente i record agli utenti, perciò le unioni di hash sono adatte per query eseguite da utenti in linea, se le tabelle sono piccole e possono essere lette velocemente.

Elaborazione di unioni esterne

Nell'elaborare un'unione esterna, l'ottimizzatore utilizza uno dei tre metodi descritti nei paragrafi precedenti. Ad esempio, se la query dell'esempio eseguisse un'unione esterna tra LAVORATORE e ALLOGGIO, potrebbe essere utilizzata un'operazione NESTED LOOPS OUTER anziché una semplice NESTED LOOPS. In un'operazione NESTED LOOPS OUTER, la tabella esterna per l'unione esterna viene normalmente utilizzata come tabella guida per la query; nel momento in cui la tabella interna viene letta alla ricerca dei record corrispondenti, vengono restituiti valori NULL per le righe che non trovano corrispondenza.

Hint

Per cambiare la selezione di un metodo di unione da parte dell'ottimizzatore, esistono alcuni hint che consentono di specificare il tipo di metodo di unione da utilizzare o l'obiettivo del metodo stesso.

Oltre agli hint descritti in questo paragrafo, si possono utilizzare le opzioni FULL e INDEX descritte precedentemente per influenzare il modo in cui le unioni vengono elaborate. Ad esempio, se si utilizza un hint per forzare l'esecuzione di un'unione NESTED LOOPS, è inoltre possibile specificare quale indice utilizzare per l'unione e a quale tabella accedere attraverso una scansione completa.

Opzioni relative agli obiettivi È possibile specificare un hint che indirizzi l'ottimizzatore a eseguire una query con un obiettivo specifico. Gli obiettivi disponibili alle unioni sono i seguenti:

| | |
|------------|--|
| ALL_ROWS | Esegue le query in modo da restituire tutte le righe il più velocemente possibile. |
| FIRST_ROWS | Esegue la query in modo da restituire la prima riga il più velocemente possibile. |

Per default l'ottimizzatore esegue una query utilizzando un percorso d'esecuzione che consenta di ridurre il tempo globale necessario per risolverla. Quindi il funzionamento di default prevede come obiettivo l'opzione ALL_ROWS. 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 di MERGE JOIN. Tuttavia, l'obiettivo ALL_ROWS non è sempre appropriato. Ad 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.

Le opzioni disponibili consentono di mimetizzare gli obiettivi: l'opzione ALL_ROWS consente all'ottimizzatore di operare una selezione tra tutte le operazioni disponibili per ridurre il tempo di processo complessivo per la query, mentre l'opzione 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 le opzioni ALL_ROWS e FIRST_ROWS solo se prima sono state analizzate le tabelle.

Nell'esempio seguente viene modificata la query ricavata dagli esempi di unioni in modo da includere l'opzione ALL_ROWS:

```
select /*+ ALL_ROWS */ LAVORATORE.Nome, ALLOGGIO.Direttore
      from LAVORATORE, ALLOGGIO
     where LAVORATORE.Alloggio = ALLOGGIO.Alloggio;
```

La stessa query può essere modificata in modo da utilizzare FIRST_ROWS:

```
select /*+ FIRST_ROWS */ LAVORATORE.Nome, ALLOGGIO.Direttore
      from LAVORATORE, ALLOGGIO
     where LAVORATORE.Alloggio = ALLOGGIO.Alloggio;
```

Con l'opzione FIRST_ROWS, l'ottimizzatore utilizzerà probabilmente meno l'operazione di MERGE JOIN a favore di NESTED LOOPS. Il metodo di unione selezionato dipende parzialmente dal resto della query. Nella query dell'esempio non è contenuta una clausola order by (che rappresenta un'operazione su insiemi eseguita tramite un SORT ORDER BY, come spiegato nel paragrafo "Operazioni che gestiscono insiemi di dati", più indietro in questo capitolo). Se la query viene corretta in modo da contenere una clausola order by, come viene mostrato nel listato che segue, in che modo il processo di unione ne risulta influenzato?

```
select /*+ FIRST_ROWS */ LAVORATORE.Nome, ALLOGGIO.Direttore
  from LAVORATORE, ALLOGGIO
 where LAVORATORE.Alloggio = ALLOGGIO.Alloggio
   order by LAVORATORE.Nome;
```

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 non viene visualizzato alcun record all'utente, fino a che tutti i record non sono stati ordinati. Quindi l'hint `FIRST_ROWS`, in questo esempio, indica all'ottimizzatore di eseguire l'unione 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 dell'hint `FIRST_ROWS` sul percorso esecutivo della query (poiché un'operazione di `SORT ORDER BY` è lenta nel restituire i record all'utente, indipendentemente dal metodo di unione utilizzato).

Hint riguardanti i metodi Oltre alla specifica degli obiettivi che l'ottimizzatore deve avere per valutare le alternative di metodi di unione, è possibile elencare le operazioni specifiche da eseguire e le tabelle sulle quali andranno a operare. Se una query coinvolge due tabelle, quando si fornisce l'opzione di hint per il metodo da utilizzare, non è necessario specificare quali tabelle unire.

L'opzione `USE_NL` indica all'ottimizzatore di utilizzare, per unire le tabelle, l'operazione `NESTED LOOPS`. Nel seguito viene specificato l'hint `USE_NL` per l'esempio relativo alla query di unione. All'interno del l'hint viene specificata la tabella `ALLOGGIO` come tabella interna per l'unione.

```
select /*+ USE_NL(ALLOGGIO) */
        LAVORATORE.Nome, ALLOGGIO.Direttore
      from LAVORATORE, ALLOGGIO
     where LAVORATORE.Alloggio = ALLOGGIO.Alloggio
       order by LAVORATORE.Nome;
```

Se si vuole fare in modo che tutte le unioni di una query su più tabelle utilizzino operazioni di `NESTED LOOPS`, è sufficiente specificare l'opzione `USE_NL` con nessun riferimento ad alcuna tabella, come viene mostrato nell'esempio che segue:

```
select /*+ USE_NL */
        LAVORATORE.Nome, ALLOGGIO.Direttore
      from LAVORATORE, ALLOGGIO
     where LAVORATORE.Alloggio = ALLOGGIO.Alloggio
       order by LAVORATORE.Nome;
```

In generale è opportuno indicare i nomi di tabelle ogni volta che si utilizza un hint che specifichi un metodo di unione, poiché non si sa in che modo la query verrà utilizzata in futuro. L'utente potrebbe non essere neppure a conoscenza di come gli oggetti del database siano attualmente impostati; ad esempio, uno degli oggetti nella propria clausola `from` potrebbe essere una vista che è stata messa a punto per utilizzare le operazioni di `MERGE JOIN`.

Se in un hint si specifica una tabella, è necessario far riferimento all'alias della stessa. Ovvero, se la propria clausola `from` fa riferimento a una tabella come:

```
from GIO.SUA_TABELLA, IO.MIA_TABELLA
```

non va specificato un hint del tipo:

```
/*+ USE_NL(10.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 degli alias e farvi riferimento nell'hint. Ad esempio, nel caso in cui una tabella venga unita a sé stessa, la clausola from potrebbe includere il testo mostrato di seguito:

```
from LAVORATORE W1, LAVORATORE W2
```

Nel listato seguente viene mostrato un hint che forza l'unione LAVORATORE-LAVORATORE a utilizzare i NESTED LOOPS, nel quale viene specificato l'alias della tabella:

```
/*+ USE_NL(w2) */
```

Come regola generale l'ottimizzatore ignora l'hint se non si utilizza la sintassi corretta. Qualsiasi hint scritto con una sintassi errata, viene trattato come un commento (poiché si trova racchiuso tra le sequenze di caratteri /* e */).

NOTA *USE_NL è un hint, non una regola. L'ottimizzatore lo può riconoscere e scegliere comunque di ignorarlo, a seconda delle informazioni statistiche disponibili al momento dell'esecuzione della query.*

Se si utilizzano le unioni NESTED LOOPS, è necessario prestare attenzione all'ordine in cui le tabelle vengono unite. L'hint ORDERED, utilizzato con le unioni NESTED LOOPS, influenza l'ordine secondo cui le tabelle vengono unite.

Quando si specifica l'hint 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 LAVORATORE, ALLOGGIO, COMPITOLAVORATORE
```

le prime due tabelle vengono unite tramite la prima unione e il risultato di quest'ultima viene unito alla terza tabella.

Poiché l'ordine delle unioni è critico per le prestazioni delle unioni NESTED LOOPS, l'hint ORDERED viene spesso utilizzato in abbinamento a USE_NL. Se si specificano degli hint per stabilire l'ordine delle unioni, è necessario assicurarsi che la distribuzione relativa dei valori all'interno delle tabelle da unire non si modifichi 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 di MERGE JOIN tra le tabelle specificate. Nel listato seguente l'hint comunica all'ottimizzatore di effettuare un'operazione di MERGE JOIN tra le tabelle LAVORATORE e ALLOGGIO:

```
select /*+ USE_MERGE(LAVORATORE,Alloggio) */
       LAVORATORE.Nome, ALLOGGIO.Direttore
```

```
from LAVORATORE, ALLOGGIO
where LAVORATORE.Alloggio = ALLOGGIO.Alloggio;
```

A partire da ORACLE7.3 è possibile specificare l'hint USE_HASH per comunicare all'ottimizzatore di considerare l'utilizzo del metodo di unione di hash. Se non viene specificata alcuna tabella, l'ottimizzatore seleziona la prima tabella da leggere nella memoria in base alle informazioni statistiche disponibili.

```
select /*+ USE_HASH */
       LAVORATORE.Nome, ALLOGGIO.Direttore
  from LAVORATORE, ALLOGGIO
 where LAVORATORE.Alloggio = ALLOGGIO.Alloggio;
```

Se si sta utilizzando il CBO e si sono precedentemente messe a punto le query per sfruttare l'ottimizzazione basata su regole, è possibile comunicare al CBO di utilizzare il metodo basato sulle regole per elaborare la propria query. L'hint RULE indica all'ottimizzatore di utilizzare l'RBO per ottimizzare la query; tutti gli altri hint all'interno della query vengono ignorati. Nell'esempio seguente, l'hint RULE viene utilizzato durante un'operazione di unione:

```
select /*+ RULE */
       LAVORATORE.Nome, ALLOGGIO.Direttore
  from LAVORATORE, ALLOGGIO
 where LAVORATORE.Alloggio = ALLOGGIO.Alloggio;
```

In generale si dovrebbe utilizzare l'hint RULE solo se le query sono state messe a punto specificamente per l'RBO. Nonostante l'hint RULE venga tuttora gestito, è consigliabile considerare in sua vece l'utilizzo del CBO per le proprie 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
```

Per il resto della sessione viene utilizzato come ottimizzatore l'RBO. Le altre impostazioni del parametro `optimizer_goal` della sessione includono COST, CHOOSE, ALL_ROWS e FIRST_ROWS.

Altri aspetti di messa a punto

Come è già stato evidenziato nella trattazione delle operazioni di NESTED LOOPS e MERGE JOIN, le operazioni differiscono nel tempo che impiegano a restituire la prima riga dalla query. Poiché 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.

Poiché le unioni NESTED LOOPS sono in grado di restituire velocemente le righe agli utenti, vengono spesso utilizzate per le query eseguite di frequente dagli utenti in linea. La loro efficienza nel restituire la prima riga, tuttavia, viene spesso influenzata da operazioni su insiemi applicate alle righe che sono state selezionate.

Ad esempio, l'aggiunta di una clausola `order by` a una query comporta un'operazione `SORT ORDER BY` alla fine del processo della query e nessuna riga viene visualizzata all'utente fino a che tutte le righe non sono state ordinate.

Come è stato già descritto nel paragrafo “Operazioni che utilizzano gli indici”, precedentemente in questo capitolo, l'utilizzo di funzioni su una colonna impedisce al database di utilizzare un indice su tale colonna durante le ricerche dei dati. È possibile sfruttare questa informazione per disabilitare in modo dinamico gli indici e influenzare il metodo di unione scelto. Ad esempio, la query seguente non specifica un'opzione di unione, ma disabilita l'utilizzo degli indici sulla colonna `Alloggio` concatenandone i valori con una stringa nulla:

```
select /*+ RULE */
       LAVORATORE.Nome, ALLOGGIO.Direttore
  from LAVORATORE, ALLOGGIO
 where LAVORATORE.Alloggio = ALLOGGIO.Alloggio;
```

La disabilitazione dinamica degli indici consente di forzare l'utilizzo di operazioni di `MERGE JOIN` anche se è attivo l'RBO (che non accetta alcun hint).

Come è già stato notato durante la trattazione dell'operazione di `NESTED LOOPS`, l'ordine delle unioni è importante quanto il metodo selezionato. Se un'unione di grandi dimensioni, o non selettiva, è la prima di una serie, la grande mole di dati restituiti influirà negativamente sulle prestazioni delle unioni successive della query e, nel complesso, sull'intera query.

A seconda degli hint, dell'obiettivo, dell'ottimizzatore e delle informazioni statistiche, l'ottimizzatore può scegliere di utilizzare un'ampia varietà di metodi di unione all'interno della stessa query. Ad esempio, l'ottimizzatore può scegliere di valutare una query di tre tavole come un'unione `NESTED LOOPS` di due tavole, seguita da un `MERGE JOIN` dell'output prodotto dall'operazione di `NESTED LOOPS` con la terza tabella. Tali combinazioni di tipi di unione 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 viene descritto nel paragrafo seguente.

36.6 Visualizzazione del percorso di esecuzione

Per visualizzare il percorso utilizzato nell'esecuzione di una query esistono due modi: l'utilizzo del comando `explain plan` oppure, a partire da ORACLE7.3, l'esecuzione del comando `set autotrace on`. Nei paragrafi seguenti vengono spiegati entrambi i comandi; nella parte restante del capitolo verrà utilizzato il comando `set autotrace on` per illustrare i percorsi di esecuzione così come vengono riportati dall'ottimizzatore.

Utilizzo di `set autotrace on`

A partire da Oracle7.3 è possibile ottenere una generazione automatica del progetto per ciascuna transazione eseguita all'interno di SQLPLUS. Il comando `set auto-`

trace on fa in modo che ciascuna query, dopo essere stata eseguita, visualizzi sia il proprio percorso seguito nell'esecuzione che informazioni di traccia ad alto livello relative al processo coinvolto nella risoluzione della query.

Per utilizzare il comando set autotrace on è necessario aver prima creato una tabella PLAN_TABLE all'interno del proprio account. La struttura di PLAN_TABLE è cambiata con ORACLE7.3 (sono state aggiunte quattro nuove colonne) e nuovamente con ORACLE8, perciò, se si è precedentemente utilizzato il comando explain plan, è necessario eliminare e ricreare la propria copia di PLAN_TABLE. I comandi mostrati nel listato seguente eliminano qualsiasi PLAN_TABLE esistente e la sostituiscono con la versione di ORACLE7.3.

NOTA *Per poter utilizzare set autotrace on, il DBA deve prima aver creato nel database il ruolo PLUSTRACE e aver garantito questo ruolo all'account dell'utente. Il ruolo PLUSTRACE consente di accedere alle viste sottostanti, nel dizionario dei dati di ORACLE, relative alle prestazioni.*

NOTA *L'esempio che segue si riferisce a \$ORACLE_HOME. Si sostituisca questo simbolo con la home directory del software di ORACLE sul proprio sistema operativo. Il file che crea la tabella PLAN_TABLE è situato nella sottodirectory /rdbms/admin sotto la home directory del software di ORACLE.*

```
drop table PLAN_TABLE;
@$ORACLE_HOME/rdbms/admin/utlxplan.sql
```

Quando si utilizza set autotrace on, vengono inseriti dei record nella tabella PLAN_TABLE allo scopo di mostrare l'ordine delle operazioni eseguite. Quando la query è stata completata, vengono visualizzati i dati. 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 focalizza l'attenzione sulla parte di output che visualizza l'ordine delle operazioni.

Se si attiva il comando set autotrace on, le informazioni sul percorso delle proprie query non vengono visualizzate 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 al contrario si è quasi sicuri che le prestazioni di una query sono accettabili, si può utilizzare set autotrace on per verificare il percorso di esecuzione.

Nell'esempio seguente viene effettuata una scansione completa della tabella ALLOGGIO. Le righe dell'output non sono state qui riportate per brevità. Al di sotto della query viene visualizzato l'ordine delle operazioni.

```
select *
  from ALLOGGIO;

Execution Plan
-----
      0   SELECT STATEMENT Optimizer=CHOOSE
      1     0  TABLE ACCESS (FULL) OF 'ALLOGGIO'
```

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 ALLOGGIO ha un’operazione chiamante (l’istruzione select stessa).

È inoltre possibile generare l’ordine delle operazioni dei comandi DML. In questo esempio viene visualizzato il percorso di esecuzione di un’istruzione delete:

```
delete *
  from ALLOGGIO;
```

Execution Plan

```
-----
0      DELETE STATEMENT Optimizer=CHOOSE
1      0      TABLE ACCESS (FULL) OF 'ALLOGGIO'
```

Il comando delete, come previsto, coinvolge la scansione completa di una tabella. Se l’utente ha analizzato le proprie tabelle, l’output della colonna “Execution Plan” visualizza anche il numero di righe di ciascuna, il costo relativo di ciascuna fase e il costo complessivo dell’operazione. Queste informazioni consentono di evidenziare le operazioni più costose per l’elaborazione della query.

Nell’esempio seguente viene eseguita una query leggermente più complessa: una query basata su indice sulla tabella ALLOGGIO, che utilizza l’indice ALLOGGIO_PK.

```
select *
  from ALLOGGIO
 where Alloggio > 'S';
```

Execution Plan

```
-----
0      SELECT STATEMENT Optimizer=CHOOSE
1      0      TABLE ACCESS (BY ROWID) OF 'ALLOGGIO'
2      1      INDEX (RANGE SCAN) OF 'ALLOGGIO_PK'
```

Nel listato precedente vi sono tre operazioni. La seconda, INDEX RANGE SCAN, procura i dati alla prima, TABLE ACCESS BY ROWID. I dati restituiti dall’operazione TABLE ACCESS BY ROWID vengono utilizzati per soddisfare la query (operazione numero 0).

L’output precedente mostra inoltre che l’ottimizzatore effettua automaticamente un rientro del margine per ciascuna fase successiva dell’esecuzione. 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 solitamente quella che viene eseguita per prima.

Nell’esempio che segue vengono unite le tabelle LAVORATORE e ALLOGGIO senza l’ausilio di indici:

```
select LAVORATORE.Nome, ALLOGGIO.Direttore
  from LAVORATORE, ALLOGGIO
 where LAVORATORE.Alloggio||'' = ALLOGGIO.Alloggio||'';
```

Execution Plan

```

0      SELECT STATEMENT Optimizer=CHOOSE
1      0      MERGE JOIN
2      1      SORT (JOIN)
3      2      TABLE ACCESS (FULL) OF 'ALLOGGIO'
4      1      SORT (JOIN)
5      4      TABLE ACCESS (FULL) OF 'LAVORATORE'

```

I rientri del listato precedente possono creare confusione, a prima vista, 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 vengono eseguite per prime, e si tratta delle due operazioni di TABLE ACCESS FULL. Successivamente vengono eseguite le due operazioni di SORT JOIN (l'ordinamento della tabella ALLOGGIO tramite l'operazione numero 2 e quella della tabella LAVORATORE per mezzo dell'operazione numero 4).

Entrambe le operazioni, i due ordinamenti, hanno come operazione chiamante MERGE JOIN. L'operazione di MERGE JOIN fornisce i dati riportandoli all'utente tramite l'istruzione select.

Nella Figura 36.13 viene mostrato il flusso delle operazioni per quanto riguarda l'esempio di MERGE JOIN. Tale flusso delle operazioni mostra le stesse informazioni di quelle fornite dall'output di set outotrace on.

Se la stessa query fosse stata eseguita come un'unione NESTED LOOPS, sarebbe stato generato un percorso d'esecuzione differente.

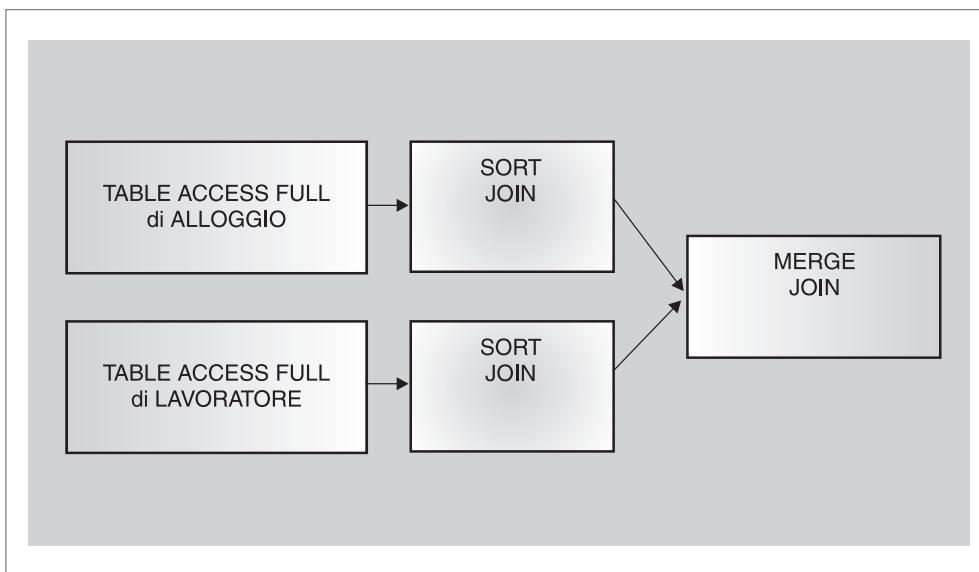


Figura 36.13 Ordine delle operazioni per l'esempio di MERGE JOIN.

Come viene mostrato nel listato seguente, l'unione NESTED LOOPS è in grado di avvantaggiarsi dell'indice ALLOGGIO_PK sulla colonna Alloggio della tabella ALLOGGIO:

```
select LAVORATORE.Nome, ALLOGGIO.Direttore
  from LAVORATORE, ALLOGGIO
 where LAVORATORE.Alloggio = ALLOGGIO.Alloggio;
```

Execution Plan

```
-----  
0      SELECT STATEMENT Optimizer=CHOOSE  
1      0      NESTED LOOPS  
2      1      TABLE ACCESS (FULL) OF 'LAVORATORE'  
3      1      TABLE ACCESS (BY ROWID) OF 'ALLOGGIO'  
4      3      INDEX (UNIQUE SCAN) OF 'ALLOGGIO_PK'
```

Le unioni NESTED LOOPS sono tra i pochi percorsi di esecuzione che non seguono la regola di “lettura dall'interno verso l'esterno”. Per leggere correttamente il percorso di esecuzione di un NESTED LOOPS, si esamini l'ordine delle operazioni che forniscono direttamente i dati all'operazione di 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). Quindi viene eseguita per prima l'operazione di TABLE ACCESS FULL della tabella LAVORATORE (LAVORATORE 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 in basso. La seconda operazione eseguita è INDEX UNIQUE SCAN dell'indice ALLOGGIO_PK. I dati ricavati dall'indice ALLOGGIO_PK vengono utilizzati per selezionare le righe dalla tabella ALLOGGIO; a questo punto, l'operazione NESTED LOOPS è in grado di restituire le righe all'utente.

Se anziché un'unione NESTED LOOPS venisse selezionata un'unione di hash il percorso di esecuzione sarebbe:

Execution Plan

```
-----  
0      SELECT STATEMENT Optimizer=CHOOSE  
1      0      HASH JOIN  
2      1      TABLE ACCESS (FULL) OF 'LAVORATORE'  
3      1      TABLE ACCESS (FULL) OF 'ALLOGGIO'
```

Il percorso di esecuzione dell'unione di hash mostra che sono state eseguite due scansioni complete delle tabelle. Poiché le due operazioni compaiono allo stesso livello di rientro, la tabella LAVORATORE viene letta per prima (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.

NOTA Se si utilizzano le opzioni per effettuare le query in parallelo o per interrogare 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 disabilitare la funzionalità di traccia automatica è sufficiente eseguire il comando set autotrace off.

Utilizzo di explain plan

Il comando explain plan può essere utilizzato per generare il percorso di esecuzione per una query senza eseguirla. Per poter utilizzare questo comando è necessario aver creato in precedenza una tabella PLAN_TABLE all'interno del proprio schema (si faccia riferimento alle istruzioni precedenti per quanto riguarda 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 processo di esecuzione prima di utilizzare una seconda volta il comando explain plan.

Nel listato che segue 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 LAVORATORE.Nome, ALLOGGIO.Direttore
  from LAVORATORE, ALLOGGIO
 where LAVORATORE.Alloggio = ALLOGGIO.Alloggio;
```

Statement processed.

Quando viene eseguito il comando explain plan, vengono inseriti dei record nella tabella PLAN_TABLE. Per interrogare la tabella PLAN_TABLE si può utilizzare la query seguente. I risultati della query mostrano le operazioni eseguite in ciascuna fase del percorso e le relazioni padre-figlio tra le fasi del percorso esecutivo, in modo analogo alla visualizzazione a rientri del comando set autotrace on:

```
select
  LPAD(' ',2*Level)||Operation||' '|Options
    ||' '|Object_Nome  Execution_Path
from PLAN_TABLE
where Statement_ID = 'TEST'
connect by prior ID = Parent_ID and Statement_ID = 'TEST'
```

```
start with ID=1;
```

La query visualizzata nel listato precedente utilizza l'operatore connect by per valutare la gerarchia delle fasi nel percorso esecutivo. Si presuppone che il campo Statment_ID sia impostato al valore 'TEST'. Le fasi del percorso esecutivo vengono visualizzate nella colonna di alias "Execution_Path".

Nel listato che segue viene mostrato l'output della query precedente:

Execution Plan

NESTED LOOPS

```
  TABLE ACCESS (FULL) OF 'LAVORATORE'
  TABLE ACCESS (BY ROWID) OF 'ALLOGGIO'
    INDEX (UNIQUE SCAN) OF 'ALLOGGIO_PK'
```

Se lo si desidera, è possibile espandere la query in modo che comprenda altre colonne della tabella PLAN_TABLE, quali ID e Parent_ID che hanno fornito rispettivamente, il numero dell'operazione e quello dell'operazione chiamante negli esempi relativi al comando set autotrace on. A partire da ORACLE7.3 è inoltre possibile selezionare la colonna Cost che visualizza il "costo" relativo di ciascuna fase.

36.7 Operazioni varie

Oltre alle operazioni mostrate precedentemente in questo capitolo, accesso alle tabelle, accesso agli indici, manipolazioni 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 a essere utilizzate meno spesso di quelle descritte precedentemente.

Filtro di righe

Quando si specifica una clausola where per eliminare righe da una query, ORACLE può utilizzare un'operazione FILTER per selezionare le righe che corrispondono con il criterio della clausola where. L'operazione FILTER non viene sempre visualizzata nell'output prodotto dal comando set autotrace on, anche se può essere stata eseguita. Si consideri la query seguente relativa alla tabella LAVORATORE:

```
select *
  from LAVORATORE
 where Nome like 'S%'
   and Eta >20;
```

Se la tabella LAVORATORE possiede un indice sulla sua colonna Nome, la query precedente viene eseguita in due fasi: un'operazione INDEX RANGE SCAN sull'indice della colonna Nome, seguita da un'operazione di TABLE ACCESS BY ROWID sulla tabella LAVORATORE (poiché tutte le colonne risultano selezionate). Quando viene applicata la condizione di limitazione seguente?

```
and Eta > 20;
```

La condizione di limitazione sulla colonna Eta viene applicata come parte di un'operazione FILTER; tuttavia, l'operazione FILTER non viene elencata in modo esplicito come parte del percorso di esecuzione. Al contrario, l'operazione FILTER viene eseguita come parte dell'operazione TABLE ACCESS BY ROWID. 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 della 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 specificata una clausola connect by, ORACLE utilizza un'operazione CONNECT BY. Il percorso di esecuzione mostra che l'operazione CONNECT BY solitamente prevede tre input nell'avanzamento verso l'alto e verso il basso all'interno di un "albero" di record. Per ottenere prestazioni migliori da una query connect by è opportuno creare un gruppo di indici che possano essere utilizzati dalle colonne specificate 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 bovino. La query seguente percorre l'albero genealogico a partire dalla radice (la mucca di nome 'EVA') per giungere ai discendenti. Può essere specificata una clausola connect by per spostarsi dai progenitori ai discendenti, o dai discendenti ai propri avi.

```
select Mucca,
       Toro,
       LPAD(' ', 6*(Level-1))||Progenie Progenie,
       Sex,
       Datanascita
  from ALLEVAMENTO
 start with Figlio = 'EVA'
connect by Mucca = PRIOR Figlio;
```

Se non esiste alcun indice sulla tabella ALLEVAMENTO, per questa query viene generato il seguente percorso esecutivo (come riportato dal comando set autotrace on):

Execution Plan

```
-----
0      SELECT STATEMENT Optimizer=CHOOSE
1      0   CONNECT BY
2      1   TABLE ACCESS (FULL) OF 'ALLEVAMENTO'
3      1   TABLE ACCESS (BY ROWID) OF 'ALLEVAMENTO'
4      1   TABLE ACCESS (FULL) OF 'ALLEVAMENTO'
```

L'operazione CONNECT BY riceve tre input: una scansione iniziale della tabella per stabilire la posizione del nodo radice, seguita da un paio di operazioni di accesso alla tabella per attraversare l'albero dei dati. Delle tre operazioni di accesso ai dati che forniscono dati all'operazione CONNECT BY, due sono operazioni di TABLE ACCESS FULL.

Per ridurre il numero di TABLE ACCESS FULL richiesti dall'operazione CONNECT BY, si dovrebbero creare un paio 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.

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 intrapreso dalla query della tabella ALLEVAMENTO utilizza ora accessi basati sugli indici, anziché scansioni complete della tabella, come viene mostrato nel listato seguente:

Execution Plan

```
0      SELECT STATEMENT Optimizer=CHOOSE
1      0      CONNECT BY
2      1      INDEX (RANGE SCAN) OF 'FIGLIO$MUCCA'
3      1      TABLE ACCESS (BY ROWID) OF 'ALLEVAMENTO'
4      1      TABLE ACCESS (BY ROWID) OF 'ALLEVAMENTO'
5      4      INDEX (RANGE SCAN) OF 'MUCCA$ALLEVAMENTO'
```

In questo listato le due operazioni di TABLE ACCESS FULL per la tabella ALLEVAMENTO sono state sostituite da accessi basati su indice. In generale gli accessi basati su indice hanno prestazioni migliori rispetto alle scansioni complete delle tabelle per le query connect by, soprattutto se vi sono più livelli nell'albero dei dati.

Query che utilizzano sequenze

Quando una query seleziona valori da una sequenza, compare nel percorso esecutivo della query l'operazione SEQUENCE. In generale i valori delle sequenze vengono selezionati interrogando DUAL (il sinonimo pubblico della tabella SYSDUAL).

Se vi è già un'altra tabella coinvolta nella query, non è necessario aggiungere DUAL alla tabella per eseguire la selezione dalla sequenza.

Ad esempio, la query seguente seleziona il successivo valore all'interno della sequenza CustomerID:

```
select CustomerID.NextVal
  from DUAL;
```

La selezione di DUAL come tabella base per la query è arbitraria; i criteri importanti sono che la tabella sia accessibile e che possieda una riga per ciascun numero della sequenza che si vuole restituire.

Per migliorare le prestazioni delle query che generano valori dalle sequenze, si può registrare nella cache un gruppo di valori preallocati in memoria della sequenza. Per default possono essere memorizzati nella cache 20 valori della sequenza. Durante questa operazione, ORACLE non riempie la cache fino a che tutti e 20 i valori sono stati utilizzati. Se il database viene disattivato, qualsiasi valore della sequenza contenuto nella cache viene perso; durante il successivo avviamento del database, verranno registrati nella cache i 20 valori successivi.

Ad esempio, è possibile alterare la sequenza CustomerID per inserire nella cache 20 valori, utilizzando il comando mostrato nel listato seguente:

```
alter sequence CustomerID cache = 20;
```

Se dalla sequenza CustomerID vengono selezionati 12 valori, otto restano ancora nella cache. Se successivamente si chiude il database, questi otto valori vengono persi. Quando si riavvia il database, viene memorizzata nella cache un'altra sequenza di 20 valori di CustomerID a partire dall'ultimo valore precedentemente inserito nella cache. Quindi vi sarà un'interruzione di otto valori della sequenza CustomerID nei valori selezionati dalla sequenza.

Query che utilizzano link di database

Se una query utilizza il link di un database 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 nei confronti di tabelle associate a cluster ha prestazioni inferiori rispetto alle operazioni effettuate su dati di tabelle non associate a cluster. Se le proprie 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 è stata utilizzata la tabella LAVORATORE come tabella guida ed è stato eseguito un accesso basato su indice per trovare i record corrispondenti nella tabella ALLOGGIO. Per accedere ai dati di ALLOGGIO è stato utilizzato l'indice ALLOGGIO_PK.

Se la tabella ALLOGGIO fosse stata memorizzata in un cluster, con la colonna Alloggio come colonna chiave del cluster, le operazioni eseguite per accedere ai dati all'interno dell'unione NESTED LOOPS sarebbero state leggermente diverse.

Se ALLOGGIO fosse in un cluster, la tabella LAVORATORE potrebbe ancora essere la tabella guida per l'unione. L'accesso ai dati della tabella ALLOGGIO potrebbe avvenire in due fasi: un'INDEX UNIQUE SCAN dell'indice della chiave del cluster, seguita da un'operazione di TABLE ACCESS CLUSTER della tabella ALLOGGIO. L'operazione di 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 ALLOGGIO si trovasse in un cluster di hash, il metodo di lettura dei dati dalla tabella cambierebbe completamente. In un cluster di hash la locazione fisica di una riga è determinata dai valori della riga stessa. Se ALLOGGIO si trova in un cluster di hash e i valori della colonna Alloggio venissero utilizzati come funzione di hash per il cluster, la query:

```
select *
  from ALLOGGIO
 where Alloggio = 'ROSE HILL';
```

potrebbe utilizzare un'operazione di TABLE ACCESS HASH per leggere i dati. Poiché la locazione fisica dei dati dovrebbe già essere nota all'ottimizzatore, non è necessario alcun accesso basato su indice.

Hint

Per comunicare all'ottimizzatore di utilizzare, durante la scansione di una tabella, un'operazione TABLE ACCESS HASH, si può specificare l'hint HASH assieme al nome (o alias) della tabella da leggere.

NOTA *Non vi è alcuna relazione tra i cluster di hash e le unioni di hash. Per specificare l'utilizzo di unioni di hash, si utilizza l'hint USE_HASH.*

Se si stanno utilizzando cluster non hash, è possibile specificare l'hint CLUSTER per comunicare che una tabella venga letta tramite un'operazione di TABLE ACCESS CLUSTER. In generale gli hint CLUSTER e HASH vengono utilizzati principalmente quando le query in questione non contengono unioni. Se una query comprende un'unione, si dovrebbero utilizzare i suggerimenti relativi alle unioni, in modo da avere impatto maggiore sul percorso esecutivo scelto per la query.

Altri aspetti legati alla messa a punto

Oltre alle due operazioni e ai relativi hint citati nei paragrafi precedenti, esistono altri due gruppi di hint che possono influenzare il processo di una query. Essi consentono di controllare l'esecuzione parallela delle query e le memorizzazioni dei dati nella cache all'interno della SGA.

Se il proprio server dispone di più processori e i dati da interrogare sono distribuiti su più periferiche è possibile migliorare il processo 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 e 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 lettura e ordinamento avviati per una query, può essere impostato a livello di tabella (si faccia riferimento al comando create table nella guida di riferimento in ordine alfabetico del Capitolo 37). 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. A partire da ORACLE7.3 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 per le risorse disponibili.

È possibile influenzare il parallelismo delle proprie query tramite l'hint PARALLEL. Nell'esempio seguente è stato impostato, per una query sulla tabella ALLOGGIO, un grado di parallelismo pari a 4:

```
select /*+ PARALLEL (Alloggio,4) */
       from ALLOGGIO;
```

Se si sta utilizzando l'opzione Parallel Server di ORACLE, è possibile specificare nell'hint PARALLEL un parametro "istanze". 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(Alloggio,4,2) */
       from ALLOGGIO
      order by Direttore;
```

Nella query precedente è stata aggiunta una seconda operazione, SORT ORDER BY per la clausola order by Direttore. Poiché anche l'operazione di ordinamento può essere eseguita in parallelo, la query può utilizzare otto processi sul server anziché quattro (quattro per la lettura della tabella e quattro per l'ordinamento). L'ottimizzatore stabilisce in modo dinamico il grado di parallelismo da utilizzare in base alle impostazioni della tabella, alla struttura del database e alle risorse del database disponibili.

Se si vuole disabilitare l'esecuzione in parallelo di una query, è sufficiente specificare l'hint NOPARALLEL. Questo hint può essere utilizzato per eseguire in modo seriale una query nei confronti di una tabella per la quale le query vengono normalmente eseguite in parallelo.

Se una tabella di piccole dimensioni (minore di cinque blocchi del database) viene letta tramite una scansione completa, i dati vengono contrassegnati per essere mantenuti il più a lungo possibile nella SGA. Anche i dati letti tramite scansioni dell'indice e operazioni di 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 dimensione maggiore di cinque blocchi e viene letta con una scansione completa, i suoi blocchi vengono solitamente contrassegnati per essere rimossi dalla SGA al più presto possibile.

Se la tabella letta con una scansione completa ha dimensione maggiore di cinque blocchi e si desidera mantenere i dati in memoria il più a lungo possibile, si può contrassegnare la tabella come “tabella da cache”. Per fare ciò si può utilizzare la clausola cache dei comandi create table o alter table.

Se una tabella di grandi dimensioni non è stata contrassegnata come tabella “da cache”, e si vuole fare in modo che i suoi dati restino nella SGA dopo che la query è stata completata, si può utilizzare l'hint CACHE per comunicare all'ottimizzatore di mantenere i dati nella SGA il più a lungo possibile. Questo hint viene solitamente utilizzato assieme a FULL. Nel listato seguente viene interrogata la tabella LAVORATORE; anche se i dati vengono letti tramite un'operazione di TABLE ACCESS FULL, non vengono immediatamente rimossi dalla SGA.

```
select /*+ FULL(LAVORATORE) CACHE(LAVORATORE) */      *
       from LAVORATORE;
```

Se una tabella viene comunemente utilizzata da molte query o utenti, e non esiste alcuno schema di indicizzazione appropriato disponibile, è consigliabile utilizzare la cache per migliorare le prestazioni relative agli accessi alla tabella. L'utilizzo della cache risulta maggiormente utile per tabelle statiche. Per disabilitare in modo dinamico le impostazioni della cache per una tabella, si può utilizzare l'hint NOCACHE.

36.8 Riepilogo

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 risuzione della query?
- Il tipo di unioni utilizzate: le tabelle sono indicizzate in modo appropriato per avvantaggiarsi delle operazioni di NESTED LOOPS?
- La dimensione delle tabelle: che dimensione hanno le tabelle da interrogare? Se vi sono più unioni, quant'è grande la mole di dati restituita da ciascuna?
- La selettività dei dati: qual è il grado di selettività degli 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 essere 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 fornite nel corso di questo capitolo per la successiva messa a punto delle operazioni o per la loro impostazione. La query così regolata sarà in grado di soddisfare gli obiettivi prefissati.

- Parte quinta
 - **Riferimento alfabetico**

Capitolo 37

Guida di riferimento in ordine alfabetico

- 37.1 Contenuti della guida di riferimento
- 37.2 Che cosa non contiene la guida di riferimento
- 37.3 Formato generale delle voci
- 37.4 Ordine dell'elenco

In questo capitolo vengono presentati i riferimenti relativi alla maggior parte dei comandi di ORACLE, le parole chiave, i prodotti, le funzionalità e le funzioni, con ampi riferimenti incrociati tra gli argomenti. Gli argomenti secondari all'interno di un riferimento possono essere trovati nell'indice analitico. 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.

37.1 Contenuti della guida di riferimento

Questa guida di riferimento in ordine alfabetico 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 riguardo 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'utilizzo corretto. Gli argomenti sono in ordine alfabetico e presentano ampi riferimenti incrociati, sia all'interno della guida di riferimento stessa, sia ai capitoli precedenti del libro.

37.2 Che cosa non contiene la guida di riferimento

Questa guida non è un'esercitazione su ORACLE, non spiega il funzionamento degli strumenti di sviluppo orientati alla grafica, poiché questi ultimi sono relativamente semplici da apprendere e utilizzare. Inoltre esiste un numero limitato di aree

il cui utilizzo è talmente specializzato e raro che si è ritenuto di non includerle in questa guida. In questi casi, il testo fa riferimento al manuale di ORACLE, o a un altro libro nel quale sia possibile trovare le informazioni necessarie.

37.3 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 delle sue componenti e un esempio di utilizzo. Una voce tipica è la seguente:

RPAD

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, funzioni; LPAD; LTRIM; RTRIM; Capitolo 6

SINTASSI RPAD(*stringa*,*lunghezza* [, 'riempimento'])

DESCRIZIONE 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 di default: lo spazio.

ESEMPIO select RPAD('CIAO ',24,'MONDO') from DUAL;

produce:

CIAO MONDOMONDOMONDOMOND

Parti che compongono ciascuna voce

La PAROLA CHIAVE solitamente 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 di ORACLE, è seguita da una breve qualifica, come (forma 1, oggetti schema), allo scopo di indicare che per quella parola chiave esiste più di una voce.

“Tipo” è un’indicazione generica della funzione della parola chiave, come “Funzione SQL”.

“Prodotti” specifica quali prodotti utilizzano la parola chiave specificata. “Tutti” significa qualsiasi prodotto in cui è consentito utilizzare il comando. Se vengono elencati prodotti specifici, significa che il comando o la parola chiave in esame si applicano solo a questi. Se soltanto alcune versioni di un prodotto possono utilizzare la parola chiave, vengono specificati i riferimenti alle versioni.

“*Vedere anche*” suggerisce altri argomenti intimamente 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 libri di riferimento che contengono dettagli che vanno al di là dell’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 non si distingue tra maiuscole e minuscole). Le variabili e i parametri delle variabili sono scritti in minuscolo *corsivo*. Questo formato indica che il termine specificato va sostituito con il valore appropriato. Quando compaiono delle parentesi tonde, esse vanno scritte esattamente dove compaiono, 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 SQL*NET o Net8 |
| <i>password</i> | Una password |
| <i>segmento</i> | Nome di un segmento |
| <i>tabella</i> | Nome di una tabella |
| <i>tablespace</i> | Nome di un 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 deve 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 conversione automatica dei dati.
- *valore* solitamente rappresenta una colonna NUMBER o un’espressione, o colonna, che può essere trattata come una colonna NUMBER dopo 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 conversione automatica dei dati.
- *intero* deve essere un numero intero, quale -3, 0, o 12.
- *espressione* sta a rappresentare qualsiasi formato di colonna. Può essere una sequenza di lettere, una variabile, un’operazione matematica, una funzione o, teoricamente, qualsiasi combinazione di funzioni e colonne il cui risultato finale sia rappresentato da un singolo valore, quale una stringa, un numero o una data.
- Occasionalmente, vengono utilizzate altre notazioni, del tipo *condizione* o *query*. Ciò viene spiegato o risulta evidente nel contesto.
- *Elementi opzionali*, che vengono racchiusi tra parentesi quadre, come in [*utente*], che significa che *utente* non è indispensabile.

- *Elementi alternativi* in un elenco vengono separati da un'unica barra verticale spezzata a metà come per OFF | ON, che va letto “OFF oppure ON”. Su alcuni sistemi questa barra verticale viene visualizzata come una barra intera.
- *Opzioni richieste*; quando è richiesto un elemento di un elenco, esso viene racchiuso tra parentesi graffe, come per {OFF | ON}.
- L'elemento di *default* 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 qualsiasi numero di colonne aggiuntive, separate tra loro da virgole.
- Raramente 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

Un numero limitato di comandi possiede 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 una applicazione 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.

37.4 Ordine dell'elenco

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).**
- segno di negazione, utilizzato su alcuni sistemi per indicare “diverso da”.
- ! **punto esclamativo.**
- ” **doppi apici.**
- # **cancelletto.**

| | |
|-------|---|
| \$ | dollaro. |
| ? | punto interrogativo. |
| % | segno di percentuale. |
| & | “e” commerciale. |
| && | 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. |
| > | maggiore di. |
| >= | maggiore o uguale a. |
| @ | chiocciola. |
| @@ | doppia chiocciola. |
| [] | parentesi quadre. |
| ^ | accento circonflesso. |
| ^= | diverso da. |
| {} | parentesi graffe. |
| | barra verticale spezzata. |
| | concatenazione. |

_ (trattino di sottolineatura)

Il trattino di sottolineatura rappresenta un'unica posizione assieme all'operatore LIKE. Vedere LIKE.

_EDITOR

Vedere EDIT.

! (punto esclamativo)

TIPO Operatore SQL, oppure operatore di ricerca del testo

PRODOTTI Tutti

VEDERE ANCHE =, CONTAINS, SOUNDEx, OPERATORI DI RICERCA TE-STUALE, Capitolo 29.

PRODOTTI All'interno di SQL il punto esclamativo ! viene utilizzato come parte dell'espressione "diverso da" !=. Ad esempio, di seguito vengono selezionate tutte le città che non appartengono al continente Asia:

```
select Citta  
      from LOCALITA  
     where Continente != 'ASIA';
```

All'interno di 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.

```
select Resume  
      from PROSPETTIVA  
     where CONTAINS(Resume, '!guarding')>0;
```

" (doppi apici)

TIPO Delimitatore SQL

PRODOTTI Tutti

VEDERE ANCHE ALIAS, TO_CHAR, Capitolo 6 e Capitolo 8

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)**TIPO** Comando SQL*PLUS**PRODOTTO** SQL*PLUS**VEDERE ANCHE** DOCUMENT, REMARK, /* */, Capitolo 5**DESCRIZIONE** # completa un blocco di documentazione in un file d'avviamento di SQLPLUS, dove il blocco inizia con la parola DOCUMENT. SQL*PLUS ignora tutte le righe a partire dalla riga con la parola DOCUMENT fino alla riga dopo il #.**\$ (dollaro)****TIPO** Comando SQL*PLUS od operatore di ricerca testuale di ConText**PRODOTTO** SQLPLUS, ConText**VEDERE ANCHE** @@, HOST, AVVIO, CONTAINS, OPERATORI DI RICERCA TESTUALE, Capitolo 29**SINTASSI** Per SQL*PLUS:

```
$ comando host
```

Per 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 un diminutivo di HOST. Non funziona per tutti i tipi di hardware o sistemi operativi. All'interno di ConText, \$ indica l'esecuzione di un'espansione a radice sul termine da ricercare. Un'espansione a radice include quelle parole che hanno la stessa radice. Ad esempio, un'espansione a radice della parola 'work' include 'works', 'working' e così via. Vedere il Capitolo 29 per l'utilizzo di queste espansioni nelle ricerche di testi.**? (punto interrogativo)****TIPO** Operatore di ricerca testuale**PRODOTTI** ConText**VEDERE ANCHE** CONTAINS, OPERATORI DI RICERCA TESTUALE, Capitolo 29

```
SINTASSI select colonna
              from tabella
             where CONTAINS(testo_colonna, '?termine')>0;
```

DESCRIZIONE All'interno di ConText, ? segnala al motore di ricerca dei testi di eseguire una ricerca a corrispondenza fuzzy. I termini da ricercare vengono espansi a includere parole che sono scritte in modo simile al termine cercato, utilizzando l'indice per i testi come fonte delle possibili corrispondenze. Vedere il Capitolo 29.**% (segno percentuale)**

% è un carattere jolly utilizzato per rappresentare qualsiasi numero di posizioni e caratteri con l'operatore LIKE. Vedere LIKE.

%FOUND

TIPO Attributo di cursore

PRODOTTI PL/SQL

VEDERE ANCHE %ISOPEN, %NOTFOUND, %ROWCOUNT, CURSOR, SQL CURSOR, Capitolo 22

SINTASSI *cursore*%FOUND

oppure:

SQL%FOUND

DESCRIZIONE %FOUND rappresenta un flag di successo per select, insert, update e delete. *cursore* 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. Viene utilizzato nella logica PL/SQL come parte di un testo IF/THEN e viene sempre valutato al valore 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.

Di seguito sono riportati gli stati di %FOUND sotto diverse condizioni:

| ATTIVITÀ DEL CURSORE | CURSOR ESPPLICITO | CURSOR IMPLICITO (SQL) |
|--|-------------------|------------------------|
| Cursore esplicito non ancora aperto | (ERROR) | n/a |
| Cursore aperto ma non ancora eseguito tramite FETCH | NULL | n/a |
| FETCH ha restituito una riga | TRUE | TRUE |
| FETCH non ha restituito alcuna riga | FALSE | FALSE |
| Prima di qualsiasi istruzione SQL eseguita in un cursore隐式 | n/a | NULL |
| insert ha inserito una riga | n/a | TRUE |
| insert ha fallito nell'inserire una riga | n/a | FALSE |
| update ha avuto effetto su almeno una riga | n/a | TRUE |
| update non ha avuto effetto su alcuna riga | n/a | FALSE |
| delete ha eliminato almeno una riga | n/a | TRUE |
| delete non ha eliminato alcuna riga | n/a | FALSE |

Effettuare il 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).

Si può notare come molte delle condizioni precedenti siano “n/a”, cioè “non applicabili”. Questo perché solo il cursore implicito viene utilizzato per qualsiasi operazione di insert, update o delete e qualsiasi controllo del suo valore deve essere effettuato utilizzando SQL%FOUND dopo la sua esecuzione e prima che qualsiasi altra istruzione SQL venga eseguita nel cursore di SQL.

%ISOPEN

TIPO Attributo di cursore

PRODOTTI PL/SQL

VEDERE ANCHE SQL CURSOR, Capitolo 22

SINTASSI *cursore*%ISOPEN

DESCRIZIONE *cursore* deve essere il nome di un cursore dichiarato esplicitamente o il cursore implicito di nome SQL. Viene valutato pari a 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 CURSOR* SQL). %ISOPEN viene utilizzato nella logica PL/SQL; non può far parte di un’istruzione SQL.

ESEMPIO ...

```
if NOT (REGISTRO%ISOPEN)
    then open REGISTRO;
end if;
...
```

%NOTFOUND

Vedere %FOUND.

%ROWCOUNT

TIPO Attributo di cursore

PRODOTTI PL/SQL

VEDERE ANCHE CLOSE, DECLARE, DELETE, FETCH, INSERT, OPEN, SELECT, UPDATE, Capitolo 22

SINTASSI *cursore*%ROWCOUNT

DESCRIZIONE *cursore* rappresenta o il nome di un cursore dichiarato in modo esplicito, oppure il cursore implicito di nome SQL. *cursore*%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 solo una riga (ad 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 più recente cursore implicito aperto. Se nessun cursore implicito è stato aperto, SQL%ROWCOUNT restituisce NULL.

%ROWTYPE

TIPO Attributo di variabile

PRODOTTI PL/SQL

VEDERE ANCHE FETCH, Capitolo 22

SINTASSI {[utente.]tabella | cursore}%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. 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. Ad esempio, si riconsideri la tabella LAVORATORE:

| Colonna | Tipodato |
|----------|-----------------------|
| Nome | VARCHAR2(25) not null |
| Eta | NUMBER |
| Alloggio | VARCHAR2(15) |

Per creare una variabile che contenga i campi corrispondenti, si utilizzi declare, il nome di una variabile e %ROWTYPE con il nome della tabella e quindi si selezioni una riga nel record (si noti il simbolo di *, che preleva tutte le colonne. Viene richiesto per la forma che utilizza il nome di una tabella come prefisso a %ROWTYPE).

```

DECLARE
  LAVORATORE_RECORD    LAVORATORE%rowtype;

BEGIN
  select * into LAVORATORE_RECORD from LAVORATORE
  where Nome = 'MARIO ROSSI';
  if LAVORATORE_RECORD.Eta > 65
    then ...
  end if;

END;
  
```

Poiché LAVORATORE_RECORD ha la stessa struttura della tabella LAVORATORE, è possibile far riferimento al campo Eta come LAVORATORE_RECORD.Eta, 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 campi singoli all'interno del record possono essere caricati utilizzando la notazione :=, come viene mostrato di seguito:

```
LAVORATORE_RECORD.Eta := 44;
```

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 deve essere dato un alias nell'istruzione select prima che si possa farvi riferimento tramite questo metodo.

Ad esempio, si supponga che l'istruzione select del cursore sia la seguente:

```
DECLARE
cursor impiegato is select Nome, Eta + 10 from LAVORATORE;
```

Non vi è alcun modo per far riferimento all'espressione Eta + 10, perciò viene ri-scritta nel modo seguente:

```
DECLARE
cursor impiegato is select Nome, (Eta + 10) Nuova_Eta
from LAVORATORE;
```

L'esempio precedente potrebbe quindi apparire:

```
DECLARE
cursor IMPIEGATO is select Nome, (Età + 10) Nuova_Eta
from LAVORATORE
where Nome = 'MARIO ROSSI';

LAVORATORE_RECORD    IMPIEGATO%rowtype;

BEGIN
open IMPIEGATO;
fetch IMPIEGATO into LAVORATORE_RECORD;

if LAVORATORE_RECORD.nuova_eta > 65
  then ...
end if;
close IMPIEGATO;

END;
```

Nuova_Eta rappresenta l'alias per Eta + 10 e ora vi si può fare riferimento come a un campo all'interno di LAVORATORE_RECORD. %ROWTYPE viene utilizzato nella logica PL/SQL; non può rappresentare una parte di un'istruzione SQL.

%TYPE

TIPO Attributo di variabile

PRODOTTI PL/SQL

VEDERE ANCHE %ROWTYPE, Capitolo 22

SINTASSI {[utente.]tabella.colonna | variabile}%TYPE

DESCRIZIONE %TYPE viene utilizzato per dichiarare una nuova variabile dello stesso tipo di quella dichiarata precedentemente, oppure come colonna particolare di una tabella già esistente nel database a cui si è collegati.

ESEMPIO In questo esempio viene assegnato a una nuova variabile, Impiegato, lo stesso tipo della colonna Nome della tabella LAVORATORE. Poiché Impiegato ora esiste, può essere utilizzata per dichiarare un'altra nuova variabile, Nuovo_Lavoratore.

```
Impiegato    LAVORATORE.Nome%TYPE;
Nuovo_Lavoratore  Impiegato%TYPE;
```

& o && (e commerciale o e commerciale doppia)

TIPO Prefissi di variabili, oppure operatore di ricerca del testo

PRODOTTO SQL*PLUS, ConText

VEDERE ANCHE :, ACCEPT, DEFINE, START, CONTAINS, OPERATORI DI RICERCA TESTUALE, Capitoli 13, 20 e 29

SINTASSI Per SQL*PLUS:

```
&intero  
&variabile  
&&variabile
```

Per ConText:

```
select colonna  
      from tabella  
     where CONTAINS(testo_colonna, 'termine & termine') > 0;
```

DESCRIZIONE & e && possono essere utilizzati in diversi modi (&& si applica solamente alla seconda definizione descritta di seguito).

- Prefisso per parametri in un file di avvio di SQL*PLUS. &1, &2 e così via vengono sostituiti da parole. Vedere START.
- Prefisso per una variabile di sostituzione in un comando SQL all'interno di 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.
- & può inoltre essere utilizzato per segnalare una condizione AND per ricerche di testi che coinvolgono più termini nell'ambito di ConText. Se anche solo uno dei termini ricercati non viene trovato, il testo non viene restituito dall'operazione di ricerca. Per questo argomento si rimanda al Capitolo 29.

' (apice singolo)

TIPO Delimitatore di stringa letterale di SQL

PRODOTTI Tutti

VEDERE ANCHE " (apice doppio), Capitolo 6

SINTASSI 'stringa'

DESCRIZIONE ' racchiude una stringa letterale, quale una stringa di caratteri o una costante di tipo data. Per specificare un apice o un apostrofo all'interno di una costante di tipo stringa, è necessario inserire due simboli ' (non un doppio apice). Conviene evitare l'utilizzo di apici nelle date (e altrove) ogni volta che è possibile.

ESEMPIO select 'Guglielmo' from DUAL;

produce l'output:

```
Guglielmo
```

mentre l'istruzione seguente:

```
select 'Guglielmo l''attore' from DUAL;
```

produce:

```
Guglielmo l'attore
```

() (parentesi)**TIPO** Delimitatore SQL**PRODOTTI** Tutti**VEDERE ANCHE** PRECEDENZA, SOTTOQUERY, UPDATE, Capitoli 3, 11 e 14**DESCRIZIONE** Delimita sottoquery, elenchi di colonne o precedenze di controlli.*** (moltiplicazione)****TIPO** Operatore SQL**PRODOTTI** Tutti**VEDERE ANCHE** +, -, /, Capitolo 7, Capitolo 29**SINTASSI** *valore1 * valore2***DESCRIZIONE** *valore1 * valore2* significa *valore1* moltiplicato per *valore2*. All'interno di ConText, * viene utilizzato per ponderare i risultati delle ricerche per termini differenti a pesi diversi. Vedere Capitolo 29.**** (esponenziale)****TIPO** Operatore PL/SQL**PRODOTTI** PL/SQL**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)****TIPO** Operatore SQL**PRODOTTI** Tutti**VEDERE ANCHE** -, *, /, Capitolo 7**SINTASSI** *valore1 + valore2***DESCRIZIONE** *valore1 + valore2* significa *valore1* più *valore2*.**- (sottrazione [forma 1])****TIPO** Operatore SQL, oppure operatore di ricerca dei testi.**PRODOTTI** Tutti**VEDERE ANCHE** +, *, /, Capitolo 7, CONTAINS, MINUS, OPERATORI DI RICERCA TESTUALE, Capitolo 29**SINTASSI** *valore1 - valore2***DESCRIZIONE** *valore1 - valore2* significa *valore1* meno *valore2*. All'interno di 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. Vedere Capitolo 29.**- (trattino [forma 2])****TIPO** Comando SQL*PLUS**PRODOTTO** SQL*PLUS**VEDERE ANCHE** Capitolo 13

SINTASSI testo del comando -
testo -
testo

DESCRIZIONE Continuazione del comando SQL*PLUS. - continua un comando sulla riga seguente.

ESEMPIO ttitle left 'Portafoglio attuale' -
right '1 Aprile 1998' skip 1 -
center 'Elenchi Industrie ' skip 4;

-- (commento)

TIPO Delimitatore di commento SQL limitato a un'unica riga

PRODOTTI Tutti

VEDERE ANCHE /* */, REMARK, Capitolo 5

SINTASSI -- *qualsiasi testo*

DESCRIZIONE -- comunica a ORACLE l'inizio di un commento. Qualsiasi cosa sia presente da questo punto fino alla fine della riga riguarda 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])

TIPO Operatore SQL

PRODOTTO SQL*PLUS

SINTASSI &*variabile.suffisso*

DESCRIZIONE Il . 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 inoltre essere utilizzata in una clausola where.

. (punto [forma 2])

TIPO Operatore SQL

PRODOTTI Tutti

VEDERE ANCHE OPERATORI SINTATTICI, Capitoli 21, 4 e 35

SINTASSI [*utente*.][*tabella*.]*colonna*

DESCRIZIONE Il . è un separatore di nomi, utilizzato per specificare il nome completo di una colonna, includendo (opzionalmente) la relativa tabella o l'utente. Vie-

ne inoltre utilizzato per specificare colonne all'interno di tipi di dati astratti nelle operazioni di select, update e delete.

ESEMPIO select Talbot.LAVORATORE.Nome
 from Talbot.LAVORATORE, Wallbom.LAVORATORE
 where Talbot.LAVORATORE.Nome = Wallbom.LAVORATORE.Nome;

.. (a)

Vedere CICLI.

/ (divisione [forma 1])

TIPO Operatore SQL

PRODOTTI Tutti

VEDERE ANCHE +, -, *, Capitolo 7

SINTASSI valore1 / valore2

DESCRIZIONE valore1 / valore2 significa valore1 diviso per valore2.

/ (barra [forma 2])

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE ;, BUFFER, EDIT, GET, RUN, SET, SQLTERMINATOR, Capitolo 13

DESCRIZIONE / esegue l'istruzione SQL nel buffer di SQL senza visualizzarla, contrariamente a RUN, che prima di eseguirla la visualizza

/* */ (commento)

TIPO Delimitatori di un commento SQL

PRODOTTI Tutti

VEDERE ANCHE REMARK, Capitolo 5

SINTASSI /* testo qualsiasi */

DESCRIZIONE /* comunica a ORACLE l'inizio di un commento. Qualsiasi elemento vi sia da quel punto in avanti, anche molte parole su più righe, si riferisce a un commento, fino a che non si incontra il delimitatore di fine commento */. Questi delimitatori vengono utilizzati solo all'interno di SQL stesso o in un programma PL/SQL e devono comparire prima del terminatore SQL. 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 */
  from GIORNALE /* questo è un altro commento */
    where Sezione = 'F'
```

Il listato che segue non è corretto, perché vi è racchiuso un commento all'interno di un altro:

```
select Argomento, Sezione, Pagina
/* questo è un commento /* con un commento racchiuso */ in esso */
  from GIORNALE
    where Sezione = 'F'
```

: (due punti, prefisso di variabile host)**TIPO** Prefisso**PRODOTTI** PL/SQL, precompilatori**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 il 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. Attualmente queste variabili devono essere composte da un singolo valore (gli array non vengono gestiti). Esse possono essere utilizzate ovunque possa essere specificata una variabile di 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 BEGIN

```
    select COUNT(*) into :ContoRighe from REGISTRO;  
END;
```

:= (uguale a)**TIPO** Operatore di assegnamento**PRODOTTI** PL/SQL**VEDERE ANCHE** DECLARE, FETCH, SELECT INTO, Capitolo 22**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;
Title := 'BARBIERI SBARBATI';
Nome := NULL**; (punto e virgola)****TIPO** Terminatore di SQL e PL/SQL, od operatore di ricerca di testi**PRODOTTI** Tutti**VEDERE ANCHE** / (Barra), BUFFER, EDIT, GET, RUN, SQLTERMINATOR, CONTAINS, NEAR, OPERATORI DI RICERCA TESTUALE, Capitolo 29

DESCRIZIONE ; esegue l'istruzione SQL o il comando che lo precede. All'interno di ConText indica di effettuare una ricerca di vicinanza per le stringhe di testo specificate. Se i termini specificati sono ‘estate’ e ‘affitto’, una ricerca di vicinanza potrebbe essere:

```
select Testo  
  from SONNET  
 where CONTAINS(Testo,'estate;affitto') >0;
```

Nel valutare i risultati della ricerca, ai testi che possiedono le parole ‘estate’ e ‘affitto’ tra di loro vicine viene attribuito un punteggio più alto rispetto ad altri in cui le due parole sono molto lontane fra loro.

< < > > (delimitatore di nome dell'etichetta PL/SQL)

Vedere BEGIN; BLOCCO, STRUTTURA; END; GOTO; CICLI; Capitolo 22.

@ (chiocciola [forma 1])

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

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 riga di comando.

@ (chiocciola [forma 2])

TIPO Operatore della sintassi SQL

PRODOTTI Tutti

VEDERE ANCHE CONNECT, COPY, LINK DI DATABASE, Capitolo 21

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)

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

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)

TIPO Operatore di ricerca di testi

PRODOTTO ConText

VEDERE ANCHE CONTAINS, OPERATORI DI RICERCA TESTUALE, Capitolo 29

SINTASSI All'interno di ConText, {} indica che il testo racchiuso venga considerato parte di una stringa di ricerca anche se si tratta di una parola riservata. Se il termine ricercato è 'in and out' e si vuole ritrovare l'intera frase, è necessario racchiudere la parola 'and' tra parentesi graffe:

```
select Testo
  from SONNET
 where CONTAINS(Testo,'in {and} out') >0;
```

| (barra verticale spezzata)

TIPO Operatore di SQL*PLUS, operatore di ricerca di testi

PRODOTTO SQL*PLUS, ConText

VEDERE ANCHE BTITLE, HEADSEP, TTITLE, OPERATORI DI RICERCA TESTUALE, Capitolo 5, Capitolo 29

SINTASSI *testo|testo*

DESCRIZIONE Quando viene utilizzato in column, ttitle o btitle di SQL*PLUS, | è il carattere headsep di default 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”). Su alcune macchine questo simbolo viene rappresentato come una barra verticale continua). All’interno di ConText, | 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 un carattere headsep.

```
TTITLE 'Questa è la prima riga|e questa è la seconda'
```

produce:

```
Questa è la prima riga  
e questa è la seconda
```

Nell’esempio seguente rappresenta il modo in cui viene utilizzato come clausola OR in una query di ricerca dei testi:

```
select colonna  
from tabella  
where CONTAINS(testo_colonna, 'termine | termine') > 0;
```

|| (concatenazione)

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE .(punto [forma 1]), CONCAT, SUBSTR, Capitolo 6

SINTASSI *espressione1 || espressione2*

DESCRIZIONE || collega o concatena due stringhe tra di loro. Il simbolo | viene chiamato barra verticale spezzata. Su alcuni computer questa barra può essere continua, anziché spezzata.

ESEMPIO Si utilizzi || per visualizzare una colonna di Citta, seguite da una virgola, uno spazio e dal paese a cui appartengono:

```
select Citta||', '||Paese from LOCALITA
```

ABS (valore assoluto)

TIPO SQL, funzioni PL/SQL

PRODOTTI Tutti

VEDERE ANCHE NUMERI FUNZIONI, Capitolo 7

SINTASSI ABS(*valore*)

valore deve 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

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE &, &&, DEFINE, Capitolo 13

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, in caso contrario ACCEPT produce un messaggio d'errore. FORMAT specifica il formato di input per la risposta (come A20). DEFAULT imposta il valore di default, 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 nasconde il valore inserito dall'utente ed è utile per password e simili.

ESEMPI Nell'esempio che segue viene visualizzato il prompt "Soprannome:" e viene registrata la voce inserita dall'utente in una variabile di tipo carattere di nome Nomignolo, sopprimendo la visualizzazione della voce inserita:

```
ACCEPT Nomignolo CHAR PROMPT 'Soprannome: ' HIDE
```

Di seguito, ACCEPT chiede all'utente "Che temperatura ha?" e registra il dato inserito dall'utente in una variabile di nome Temperatura:

```
ACCEPT Temperatura NUMBER PROMPT 'Che temperatura ha? '
```

ACCUMULATE

Vedere OPERATORI DI RICERCA TESTUALE.

ACOS

TIPO Comando SQL

PRODOTTI Tutti

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 devono essere compresi tra -1 e 1; l'output è espresso in radianti.

ADD_MONTHS

TIPO SQL, funzione PL/SQL

PRODOTTI Tutti

VEDERE ANCHE DATE, FUNZIONI; Capitolo 8

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* deve 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 è stato eseguito il 1° Aprile, 1998 alle 11 P.M.

```
select ADD_MONTHS(SysDate,1), ADD_MONTHS(SysDate,.3) from DUAL;
```

```
ADD_MONTH ADD_MONTH
```

```
----- -----
```

```
01-MAY-98 01-APR-98
```

ADDRESS (ROW)

Vedere ROWID.

ALIAS

L'alias rappresenta un nome temporaneo assegnato a una tabella o a 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 di SQL*PLUS (se si tratta di una colonna). Per separare la definizione della colonna dal proprio alias, si può utilizzare la parola chiave AS. Vedere " (apici doppi), AS e SELECT.

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 10.

ALL

TIPO Operatore logico SQL

PRODOTTI Tutti

VEDERE ANCHE ANY, BETWEEN, EXISTS, IN, OPERATORI LOGICI, Capitolo 11

SINTASSI *operatore ALL elenco*

DESCRIZIONE != ALL equivale a NOT IN. *operatore* può essere uno dei seguenti operatori: =, >, >=, <, <=, !=. *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 sotto-query diventa un membro dell'elenco, come per:

```
LOCALITA.Citta != ALL (select Citta from TEMPO)
```

Può anche essere una sequenza 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 sottoquery, come:

Prospetto != ALL (select Venditore, Cliente from . . .)

Molte persone 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 sottoquery 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 al più grande elemento 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. |
| Pag <= ALL (4,2,7) | Pag è minore o uguale al più piccolo elemento 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. |

ESEMPI Si consideri questa tabella

```
select * from GIORNALE
order by Pag, Sezione;
```

| ARGOMENTO | S | PAG |
|--------------|---|-----|
| Notizie | A | 1 |
| Vita moderna | B | 1 |
| Sport | D | 1 |
| Economia | E | 1 |
| Bridge | B | 2 |
| Meteo | C | 2 |
| Film | B | 4 |
| Fumetti | C | 4 |
| Necrologi | F | 6 |
| Medicina | F | 6 |
| Televisione | B | 7 |
| Nascite | F | 7 |
| Annunci | F | 8 |
| Editoriali | A | 12 |

La logica di ALL, utilizzata sulla tabella precedente, viene mostrata nelle query che seguono. Si tenga presente che l'elenco dei valori (4,2,7) potrebbe anche essere semplicemente una sottoquery:

```
select * from GIORNALE  
where Pag = ALL (4,2,7);
```

no rows selected

Si osservi, comunque, ciò che accade quando una sottoquery produce solo righe con `Pag = 1`:

```
select * from GIORNALE  
where Pag = ALL (select Pag from GIORNALE  
                  where Sezione IN ('D','E'));
```

| ARGOMENTO | S | PAG |
|--------------|---|-----|
| Notizie | A | 1 |
| Sport | D | 1 |
| Economia | E | 1 |
| Vita moderna | B | 1 |

Compaiono anche le sezioni A e B. Di seguito vengono riportati altri esempi:

```
select * from GIORNALE  
where Pag > ALL (4,2,7);
```

| ARGOMENTO | S | PAG |
|------------|---|-----|
| Editoriali | A | 12 |
| Annunci | F | 8 |

```
select * from GIORNALE  
where Pag >= ALL (4,2,7);
```

| ARGOMENTO | S | PAG |
|-------------|---|-----|
| Editoriali | A | 12 |
| Televisione | B | 7 |
| Nascite | F | 7 |
| Annunci | F | 8 |

```
select * from GIORNALE  
where Pag < ALL (4,2,7);
```

| ARGOMENTO | S | PAG |
|--------------|---|-----|
| Notizie | A | 1 |
| Sport | D | 1 |
| Economia | E | 1 |
| Vita moderna | B | 1 |

```
select * from GIORNALE
where Pag <= ALL (4,2,7);
```

| ARGOMENTO | S | PAG |
|--------------|---|-----|
| Notizie | A | 1 |
| Sport | D | 1 |
| Economia | E | 1 |
| Meteo | C | 2 |
| Vita moderna | B | 1 |
| Bridge | B | 2 |

```
select * from GIORNALE
where Pag != ALL (4,2,7);
```

| ARGOMENTO | S | PAG |
|--------------|---|-----|
| Notizie | A | 1 |
| Sport | D | 1 |
| Editoriali | A | 12 |
| Economia | E | 1 |
| Annunci | F | 8 |
| Vita moderna | B | 1 |
| Necrologi | F | 6 |
| Salute | F | 6 |

ALLOCATE (interno SQL)

TIPO Comando SQL dichiarativo per il precompilatore

PRODOTTO Precompilatori

VEDERE ANCHE DECLARE CURSOR

SINTASSI EXEC SQL ALLOCATE *variabile_cursore*

DESCRIZIONE La clausola ALLOCATE alloca una variabile cursore, a cui si possa fare riferimento in un blocco PL/SQL.

ALTER CLUSTER

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE CLUSTER, CREATE TABLE, STORAGE, Capitolo 19

SINTASSI ALTER CLUSTER [*utente.*] *cluster*

```
{ PCTUSED intero
  | PCTFREE intero
  | SIZE intero [K | M]
  | INITTRANS intero
  | MAXTRANS intero
  | STORAGE registra
  | ALLOCATE EXTENT [{( SIZE intero [K | M]
    | DATAFILE 'nomefile'
    | INSTANCE intero} ...}]
  | DEALLOCATE UNUSED [KEEP intero [K | M]]} ...
  | PARALLEL clausola_parallel | NOPARALLEL]
```

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, fatta eccezione per il fatto che vengono utilizzati per modificare un cluster. SIZE viene descritto alla voce CREATE CLUSTER. Per modificare un cluster, è necessario possederlo 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.

Quando vengono letti dei blocchi dal database, ORACLE li memorizza nella SGA. ORACLE gestisce un elenco (LRU) dei blocchi utilizzati meno di recente; quando nella SGA viene richiesto spazio aggiuntivo, questi blocchi di dati utilizzati non di recente vengono rimossi dalla SGA.

CACHE si sovrappone a questo funzionamento specificando che i blocchi di quel cluster devono essere letti in modo da essere posizionati all'estremità “dei blocchi utilizzati più di recente” nell’elenco LRU. Quindi i blocchi risiedono a lungo nella SGA. Questa opzione è utile se il cluster contiene tabelle di ricerca di piccole dimensioni. NOCACHE (l’opzione di default) inverte il funzionamento del cluster, riportandolo al funzionamento dell’elenco LRU normale.

PARALLEL, assieme a DEGREE e INSTANCES, specifica le caratteristiche parallele del cluster (per i database che utilizzano l’opzione di query in parallelo). DEGREE specifica il numero di server da utilizzare per le query; INSTANCE specifica il modo in cui il cluster viene suddiviso tra le istanze di un server parallelo per l’elaborazione di query in parallelo. Un intero *n* indica che il cluster deve essere suddiviso tra il numero specificato di istanze disponibili.

ESEMPIO alter cluster COMPITOLAVORATORE size 1024 storage (maxextents 30);

ALTER DATABASE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER ROLLBACK SEGMENT, AVVIO, CHIUSURA, CREATE DATABASE

SINTASSI ALTER DATABASE [*database*]

```

{ MOUNT [STANDBY DATABASE] [EXCLUSIVE | PARALLEL]
| CONVERT
| OPEN [RESETLOGS | NORESETLOGS]
| ACTIVATE STANDBY DATABASE
| ARCHIVELOG
| NOARCHIVELOG
| RECOVER clausola recover
| ADD LOGFILE [THREAD intero] [GROUP intero] filespec
          [, [GROUP INTEGER] filespec] ...
| ADD LOGFILE MEMBER 'nomefile' [REUSE]
          [, 'nomefile' [REUSE] ] ...
TO { GROUP intero
     | ('nomefile' [, 'nomefile') ...)
```

```

    | 'nomefile'}
    | [, 'nomefile' [REUSE] [, [REUSE] ] ...]
  TO { GROUP intero
    | ('nomefile' [, 'nomefile'] ...)
    | 'nomefile' } ] ...
  | DROP LOGFILE
    { GROUP intero
      | ('nomefile' [, 'nomefile'] ...)
      | 'nomefile'
        [, { GROUP intero
          | ('nomefile' [, 'nomefile'] ...)
          | 'nomefile' } ] ...
  | DROP LOGFILE MEMBER 'nomefile' [, 'nomefile'] ...
  | CLEAR [UNARCHIVED] LOGFILE
    { GROUP intero
      | ('nomefile' [, 'nomefile'] ...)
      | 'nomefile'
        [, { GROUP intero
          | ('nomefile' [, 'nomefile'] ...)
          | 'nomefile' } ] ...
    [UNRECOVERABLE DATAFILE]
  | RENAME FILE 'nomefile' [, 'nomefile'] ...
    TO 'nomefile' [, 'nomefile'] ...
  | CREATE STANDBY CONTROLFILE AS 'nomefile' [REUSE]
  | BACKUP CONTROLFILE { TO 'nomefile' [REUSE]
    | | TO TRACE [RESETLOGS | NORESETLOGS] }
  | RENAME GLOBAL_NAME TO database [.dominio] ...
  | RESET COMPATIBILITY
  | SET { DBLOW = 'testo'
    | DBHIGH = 'testo'
    | DBMAC {ON | OFF} }
  | ENABLE [PUBLIC] THREAD intero
  | DISABLE THREAD intero
  | CREATE DATAFILE 'nomefile' [, nomefile] ...
    | [AS filespec [, filespec] ...]
  | DATAFILE 'nomefile' ['nomefile'] ...
    { ONLINE
    | OFFLINE [DROP]
    | RESIZE intero [K | M]
  | AUTOEXTEND { OFF
    | | ON [NEXT intero [K | M] ]
      | | MAXSIZE {UNLIMITED | intero [K | M] } }
    | END BACKUP} }

```

Per poter modificare un database è necessario possedere il privilegio di sistema ALTER DATABASE.

database rappresenta il nome del database, di lunghezza massima pari a 8 caratteri. DB_NAME in init.ora contiene il nome del database di default. Se il database non viene specificato, viene utilizzato per default il nome DB_NAME. ADD LOGFILE aggiunge un file o più file redo log al database. *filespec* specifica i nomi dei LOGFILE e le dimensioni:

'file' [SIZE intero [K | M] [REUSE]

SIZE rappresenta il numero di byte riservati per il 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 se non esiste, nel caso esista già 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 un insieme 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.

RENAME cambia il nome di un database o di un file di log esistente.

ARCHIVELOG e **NOARCHIVELOG** definiscono il modo in cui vengono utilizzati i file redo log. **NOARCHIVELOG** è l'opzione di default e significa che i file di redo possono essere riutilizzati senza che i relativi contenuti vengano salvati da un'altra parte. In questo modo, si assicura un recupero dell'istanza, fatta eccezione per problemi di funzionamento verificatisi con i supporti, quale un problema sul disco fisso.

ARCHIVELOG forza l'archiviazione dei file di redo log (solitamente su un altro disco o nastro), in modo da consentire il ripristino dei dati a seguito di problemi di funzionamento. Questa modalità gestisce inoltre il ripristino delle istanze.

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 Parallel Server 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**.

È possibile abilitare (**ENABLE**) o disabilitare (**DISABLE**) un **THREAD**. **PUBLIC** rende il thread disponibile per qualsiasi istanza che non richieda un thread specifico.

Si può effettuare il **BACKUP** di un **CONTROLFILE** in un file specificato.

La clausola **DATAFILE** consente di mettere un file in linea (opzione **ONLINE**) o sconnetterlo (opzione **OFFLINE**). 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.

L'opzione **CREATE** consente di rimpiazzare 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.

È possibile utilizzare RENAME GLOBAL_NAME per modificare il nome del database. Se si specifica un dominio, a ORACLE viene comunicata la posizione del database all'interno della rete. È necessario modificare i riferimenti al database anche dai database remoti.

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.

L'opzione RESET COMPATIBILITY, abbinata al parametro COMPATIBLE del file init.ora, viene utilizzata per tornare a una precedente versione di ORACLE. Le opzioni possibili sono SET DBMAC ON o OFF per Trusted ORACLE oppure SET DBHIGH o DBLOW applicata a una stringa con la specifica del sistema operativo.

ESEMPIO alter database add logfile group 1 'biglog.006' size 1m reuse;

ALTER FUNCTION

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE FUNCTION, DROP FUNCTION, Capitolo 24

SINTASSI ALTER FUNCTION [*utente.*]*funzione* COMPILE

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

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE INDEX, DROP INDEX, STORAGE, Capitoli 17 e 19

SINTASSI ALTER INDEX [*utente.*] *indice*

```
[REBUILD [ {PARALLEL clausola_parallelia | NOPARALLEL}
           | {LOGGING | NOLOGGING}
           | {REVERSE | NOREVERSE}
           | clausola_attributi_fisici
           | TABLESPACE tablespace] ...]
[ DEALLOCATE UNUSED [KEEP intero [K | M] ]
| ALLOCATE EXTENT [ ( { SIZE intero [K | M]
                         | DATAFILE 'nomefile'
                         | INSTANCE intero} ...) ]
| {PARALLEL clausola_parallelia | NOPARALLEL}
| clausola_attributi_fisici
| {LOGGING | NOLOGGING}
| RENAME TO nome_nuovo_indice
| MODIFY PARTITION nome_partitione [ clausola_attributi_fisici
                                         | {LOGGING | NOLOGGING}
                                         | UNUSABLE]
| RENAME PARTITION nome_partitione TO nome_nuova_partitione
| DROP PARTITION nome_partitione
| SPLIT PARTITION vecchio_nome_partitione AT (elenco_valori)
```

```

[ INTO ( PARTITION [div_partizione_1]
        [ clausola_attributi_fisici
        | TABLESPACE TABLESPACE
        | {LOGGING | NOLOGGING} ... ]
      PARTITION [DIV_PARTIZIONE_2]
        [ clausola_attributi_fisici
        | TABLESPACE TABLESPACE
        | {LOGGING | NOLOGGING} ... ] ) ]
  [ PARALLEL clausola_parallelia | NOPARALLEL ]
  | REBUILD PARTITION nome_partizione [ clausola_attributi_fisici
    | TABLESPACE TABLESPACE
    | {PARALLEL clausola_parallelia
    | NOPARALLEL}
      | {LOGGING | NOLOGGING} ... ]
  | UNUSABLE
clausola_attributi_fisici ::==
  [ PCTFREE intero
  | INITTRANS intero
  | MAXTRANS intero
  | STORAGE regista] ...

```

DESCRIZIONE *utente* rappresenta l'utente in possesso dell'indice da modificare (gli altri utenti, per eseguire questo comando, devono possedere i privilegi DBA). *indice* rappresenta il nome di un indice esistente. Vedere CREATE TABLE per una descrizione di INITTRANS e MAXTRANS. Il valore di default per INITTRANS per quanto riguarda gli indici è 2; MAXTRANS è 255. STORAGE contiene clausole secondarie, ciascuna delle quali è descritta sotto STORAGE. Oltre ai privilegi necessari per modificare l'indice della tabella, è necessario inoltre possedere l'indice in questione oppure il privilegio di sistema ALTER 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 spazio), TABLESPACE (per spostare l'indice in un nuovo tablespace) e INITTRANS (per modificare il numero iniziale di voci).

È possibile modificare, rinominare, eliminare, suddividere o ricostruire partizioni dell'indice. Vedere il Capitolo 17 per una trattazione di indici suddivisi in partizioni.

ALTER PACKAGE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE PACKAGE, DROP PACKAGE, Capitolo 24

SINTASSI ALTER PACKAGE [*utente*.]package COMPILE [PACKAGE | BODY]

DESCRIZIONE ALTER PACKAGE ricompila le specifiche 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

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE PROCEDURE, DROP PROCEDURE, Capitolo 24

SINTASSI alter procedure [*utente.*]*PROCEDURA* compile

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

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE PROFILE, DROP PROFILE, Capitolo 18

SINTASSI ALTER PROFILE *profilo* LIMIT

```
{ { SESSIONS_PER_USER
  | CPU_PER_SESSION
  | CPU_PER_CALL
  | CONNECT_TIME
  | IDLE_TIME
  | LOGICAL_READS_PER_SESSION
  | LOGICAL_READS_PER_CALL
  | COMPOSITE_LIMIT}
  { intero | UNLIMITED | DEFAULT}
  | { PRIVATE_SGA { intero [K | M]
    | UNLIMITED
    | DEFAULT}
    | FAILED_LOGIN_ATTEMPTS
    | PASSWORD_LIFETIME
    | {PASSWORD_REUSE_TIME
      | PASSWORD_REUSE_MAX}
    | ACCOUNT_LOCK_TIME
    | PASSWORD_GRACE_TIME}
    { intero | UNLIMITED | DEFAULT}
    | PASSWORD_VERIFY_FUNCTION
    {funzione | NULL | DEFAULT} } ... }
```

DESCRIZIONE ALTER PROFILE consente di modificare l'impostazione di un profilo. Vedere CREATE PROFILE per ulteriori dettagli.

ALTER RESOURCE COST

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE PROFILE

SINTASSI ALTER RESOURCE COST

```
{CPU_PER_SESSION intero |
CONNECT_TIME intero |
LOGICAL_READS_PER_SESSION intero |
PRIVATE_SGA intero}
```

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 che dal disco durante una sessione. PRIVATE_SGA è il numero di byte contenuti nella SGA, importante solo se si sta utilizzando il server in modalità multithreading. Assegnando dei pesi, è possibile modificare la formula utilizzata per calcolare il costo della risorsa totale.

ALTER ROLE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE ROLE, DROP ROLE, Capitolo 18

SINTASSI ALTER ROLE *ruolo*

```
{ NOT IDENTIFIED  
| IDENTIFIED {BY password | EXTERNALLY | GLOBALLY} }
```

DESCRIZIONE ALTER ROLE consente di modificare un ruolo. Vedere CREATE ROLE e il Capitolo 18 per ulteriori dettagli.

ALTER ROLLBACK SEGMENT

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE DATABASE, CREATE ROLLBACK SEGMENT, CREATE TABLESPACE, DROP ROLLBACK SEGMENT, STORAGE

SINTASSI ALTER ROLLBACK SEGMENT *segmento*

```
{STORAGE registra]  
[ONLINE | OFFLINE]  
[SHRINK [TO intero [K | M]]}
```

DESCRIZIONE *segmento* è il nome assegnato al segmento di rollback. STORAGE contiene delle clausole secondarie, ciascuna delle quali viene descritta 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

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE AUDIT, CREATE SEQUENCE, DROP SEQUENCE, GRANT, NEXTVAL, REVOKE e CURRVAL alla voce PSEUDOCOLONNE, Capitolo 19

SINTASSI ALTER SEQUENCE [*utente.*]*sequenza*

```
[INCREMENT BY intero]  
[MAXVALUE intero | NOMAXVALUE]  
[MINVALUE intero | NOMINVALUE]  
[CYCLE | NOCYCLE]  
[CACHE intero | NOCACHE]  
[ORDER | NOORDER]
```

DESCRIZIONE Tutte queste opzioni influenzano l'utilizzo futuro di una sequenza esistente (si noti che start with non è disponibile; per modificare il valore con cui viene avviata una sequenza, è necessario eseguirne il DROP e ricrearla tramite CREATE). *sequenza* è il nome assegnato alla sequenza stessa. Il valore di default dell'incremento (INCREMENT BY) è pari a 1. Un numero positivo provoca un incremento in ordine crescente del numero della sequenza di un intervallo uguale all'intero. Un numero negativo causa un incremento in ordine decrescente (decremento) allo stesso modo.

MINVALUE rappresenta il numero più basso generato dalla sequenza. Il default è 1 per le sequenze in ordine crescente. MAXVALUE è invece il numero più elevato generato dalla sequenza. Per sequenze in ordine decrescente, il default è -1. Per consentire alle sequenze di proseguire senza limitazioni, è sufficiente specificare soltanto MINVALUE per quelle in ordine crescente o MAXVALUE per le sequenze in ordine decrescente. Per porre fine alla creazioni di numeri sequenziali e forzare la generazione di un errore nel caso venga fatto un tentativo di ottenere un nuovo numero, è sufficiente specificare un valore per MAXVALUE per le sequenze in ordine crescente o un valore per MINVALUE per quelle decrescenti, più NOCYCLE. Per riavviare ciascun tipo di sequenza dal numero corrispondente al proprio MAXVALUE o MINVALUE occorre specificare CYCLE. Non può essere specificato un nuovo MAXVALUE inferiore all'attuale valore di CURRVAL per le sequenze in ordine crescente. Analogamente, la stessa regola vale per MINVALUE per quanto riguarda le sequenze in ordine decrescente.

CACHE consente di registrare in memoria un gruppo prestabilito di numeri sequenziali. Il default è 20. Il valore impostato deve essere inferiore a MAXVALUE meno MINVALUE. ORDER garantisce che i numeri della sequenza vengano assegnati alle istanze che li richiedono nell'ordine in cui le richieste vengono ricevute. Ciò risulta particolarmente utile in applicazioni che richiedono un resoconto della sequenza con cui hanno avuto luogo le transazioni.

Per modificare una sequenza, è necessario disporre dei privilegi ALTER sulla sequenza stessa oppure essere in possesso del privilegio di sistema ALTER ANY SEQUENCE.

ALTER SESSION

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE *ORACLE Server Performance Tuning Guide*, *Trusted ORACLE Administrator's Guide*, *ORACLE Server Administrator's Guide*

SINTASSI ALTER SESSION

```
{SET
  [SQL_TRACE {TRUE | FALSE}]
  [GLOBAL_NAMES {TRUE | FALSE}]
  [NLS_LANGUAGE = linguaggio]
  [NLS_TERRITORY = territorio]
  [NLS_DATE_FORMAT = formato]
  [NLS_DATE_LANGUAGE = linguaggio]
  [NLS_NUMERIC_CHARACTERS = stringa]
  [NLS_ISO_CURRENCY = territorio]
  [NLS_CURRENCY = stringa]}]
```

```
[NLS_SORT = sort]
[LABEL = {stringa | DBHIGH | DBLOW | OSLABEL}]
[MLS_LABEL_FORMAT = formato] |
[OPTIMIZER_GOAL = {ALL_ROWS|FIRST_ROWS|RULE|CHOOSE}] |
[FLAGGER = [ENTRY | INTERMEDIATE | FULL | OFF]]
[SESSION_CACHED_CURSORS = intero]
CLOSE_CACHED_OPEN_CURSORS = [TRUE | FALSE]
INSTANCE = intero
HASH_JOIN_ENABLED = [TRUE | FALSE]
HASH_AREA_SIZE = intero
HASH_MULTIBLOCK_IO_COUNT = intero
REMOTE_DEPENDENCIES_MODE = [TIMESTAMP | SIGNATURE]
ISOLATION_LEVEL [SERIALIZABLE | READ COMMITTED]
CLOSE DATABASE LINK link |
ADVISE {COMMIT | ROLLBACK | NOTHING} |
{ENABLE | DISABLE} COMMIT IN PROCEDURE
| {ENABLE | DISABLE} PARALLEL DML}
```

DESCRIZIONE 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 SQL_TRACE di init.ora imposta il valore iniziale per questo report.

GLOBAL_NAMES controlla se forzare oppure no la risoluzione globale dei nomi all'interno della sessione. Il parametro GLOBAL_NAME di init.ora imposta il valore iniziale per la sessione dell'utente.

Le varie opzioni NLS consentono di impostare il supporto al linguaggio nazionale per una sessione specifica, se si possiede il privilegio di sistema ALTER SESSION. NLS_LANGUAGE specifica il linguaggio da utilizzare per gli errori e gli altri messaggi e controlla il linguaggio per i nomi dei mesi e dei giorni nei meccanismi di ordinamento. NLS_TERRITORY imposta il formato della data, i caratteri decimali, il separatore per i gruppi, i simboli della moneta locale e ISO, il primo giorno della settimana per il formato di data D e il calcolo dei numeri delle settimane ISO nelle funzioni per le date. NLS_DATE_FORMAT imposta il formato di default per la data; NLS_DATE_LANGUAGE controlla i nomi dei giorni e dei mesi; NLS_NUMERIC_CHARACTERS assegna al carattere decimale e al separatore di gruppo il formato 'dg', dove d è il carattere decimale e g il separatore di gruppo; NLS_ISO_CURRENCY assegna un territorio a un simbolo di moneta; NLS_CURRENCY assegna un simbolo di moneta specifico; infine, NLS_SORT cambia la sequenza di ordinamento di tipo linguistico. LABEL e MLS_LABEL_FORMAT fanno parte di Trusted ORACLE.

OPTIMIZER_GOAL comunica all'ottimizzatore del database quale obiettivo adottare per la propria sessione: basato su regole (RULE), basato sui costi per ottenere le migliori prestazioni di trasferimento (ALL_ROWS), basato sui costi per ottenere un miglior tempo di risposta (FIRST_ROWS) oppure basato sui costi se applicabile (CHOOSE).

FLAGGER imposta il livello FIPS SQL92; livello iniziale (Entry), intermedio (Intermediate), completo (Full) oppure disabilitato (Off).

SESSION_CACHED_CURSORS specifica il numero di cursori da registrare nella cache della propria sessione.

CLOSE_CACHED_OPEN_CURSORS controlla se i cursori PL/SQL aperti dalla propria sessione devono essere chiusi a ciascun commit. INSTANCE definisce l'istanza da utilizzare per le connessioni Parallel Server di ORACLE.

HASH_JOIN_ENABLED specifica se si possono utilizzare le unioni hash durante la sessione. In caso affermativo, l'area di memoria disponibile viene impostata tramite HASH_AREA_SIZE, mentre HASH_MULTIBLOCK_READ_COUNT comunica al database quanti blocchi alla volta leggere durante le operazioni di unione hash. REMOTE_DEPENDENCIES_MODE comunica a ORACLE il modo in cui gestire più transazioni DML (il default è READ COMMITTED, il funzionamento normale; SERIALIZABLE è disponibile in conformità a SQL92).

Per eliminare il collegamento di una sessione a un database remoto, è possibile chiudere (CLOSE) il collegamento al database (DATABASE LINK). È possibile dare comunicazione di ciò alle transazioni distribuite all'interno di database remoti tramite ADVISE. È possibile specificare se le procedure possano effettuare o meno un COMMIT.

ALTER SNAPSHOT

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE SNAPSHOT, DROP SNAPSHOT, Capitolo 28

SINTASSI ALTER SNAPSHOT [*utente.*]*snapshot*

```
{PCTFREE intero |
 PCTUSED intero |
 INITTRANS intero |
 MAXTRANS intero |
 STORAGE registra]
 [USING INDEX
  [PCTFREE intero]
  [INITTRANS intero]
  [MAXTRANS intero]
  [STORAGE registra]]
 [REFRESH {FAST | COMPLETE | FORCE}]
 [START WITH data] [NEXT data] [WITH PRIMARY KEY]}
```

DESCRIZIONE ALTER SNAPSHOT consente di modificare le caratteristiche di memorizzazione o la modalità e il tempo con cui viene rinnovato uno snapshot. Per modificare uno snapshot è necessario esserne il proprietario o possedere il privilegio di sistema ALTER ANY SNAPSHOT. Vedere CREATE SNAPSHOT e il Capitolo 28 per ulteriori dettagli.

ALTER SNAPSHOT LOG

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE SNAPSHOT, CREATE SNAPSHOT LOG, DROP SNAPSHOT LOG, Capitolo 28

SINTASSI ALTER SNAPSHOT LOG ON [*utente.*]*tabella*

```
[ ADD {[PRIMARY KEY] [,ROWID] [,,(colonna_filtro)
   | [,,(colonna_filtro)] ...}
 [ PCTFREE intero
```

```

| PCTUSED intero
| INITTRANS intero
| MAXTRANS intero
| STORAGE registra]
```

DESCRIZIONE ALTER SNAPSHOT LOG consente di modificare le caratteristiche di memorizzazione della tabella di log di uno snapshot. Per modificare il log di uno snapshot, è necessario possedere il privilegio ALTER sulla tabella di log dello snapshot oppure il privilegio di sistema ALTER ANY TABLE. Vedere CREATE SNAPSHOT LOG e il Capitolo 28.

ALTER SYSTEM

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER SESSION, ARCHIVE_LOG, CREATE PROFILE

SINTASSI ALTER SYSTEM

```

{SET {RESOURCE_LIMIT = {TRUE | FALSE} |
      MTS_SERVERS = intero |
      MTS_DISPATCHERS = 'protocollo, intero'} |
      GLOBAL_NAMES={TRUE|FALSE}
      LICENSE_MAX_SESSIONS=intero
      LICENSE_SESSIONS_WARNING=intero
      LICENSE_MAX_USERS=intero
      SWITCH LOGFILE |
      {CHECKPOINT | CHECK DATAFILES} {GLOBAL | LOCAL} |
      {ENABLE | DISABLE}
          {DISTRIBUTED RECOVERY | RESTRICTED SESSION} |
      ARCHIVE LOG archivio_log |
      FLUSH SHARED_POOL |
      KILL SESSION 'intero1, intero2'}
```

DESCRIZIONE ALTER SYSTEM consente di modificare la propria istanza di ORACLE in uno dei modi elencati di seguito.

RESOURCE_LIMIT abilita o disabilita i limiti delle risorse.

MTS_SERVERS modifica il numero di processi condivisi del server.

MTS_DISPATCHERS imposta un nuovo numero di processi dispatcher utilizzando il protocollo di rete per i processi.

GLOBAL_NAMES specifica se forzare la nomenclatura globale per il database.

LICENSE_MAX_SESSIONS, LICENSE_SESSIONS_WARNING e LICENSE_MAX_USERS consentono di impostare il limite di soglia e il limite massimo relativo al numero di sessioni e utenti ammessi nel proprio database.

SWITCH LOGFILE modifica 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 ristretta, accessibile solo agli utenti in possesso di quel privilegio di sistema.

ENABLE o DISABLE DISTRIBUTED RECOVERY abilita o disabilita, rispettivamente, questa opzione.

ARCHIVE LOG archivia manualmente i file redo log oppure abilita l'archiviazione automatica. Vedere ARCHIVE_LOG.

FLUSH SHARED_POOL cancella tutti i dati dalla risorsa condivisa della SGA.

KILL SESSION termina una sessione con la colonna SID o la colonna SERIAL# della tabella V\$SESSION come identificativi di quella sessione.

ALTER TABLE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE TABLE, DISABLE, DROP, ENABLE, Capitolo 17 e Capitolo 19

SINTASSI ALTER TABLE [*utente.*] *tabella*

```

{ADD ( [ {tipodato colonna [DEFAULT espr] [WITH ROWID]
          [SCOPE IS [utente.] nome_tabella_ambito]
          [vincolo_colonna] ...
          | vincolo_tabella | REF (nome_colonna_rif) WITH ROWID
          | SCOPE FOR (nome_colonna_rif) IS
            [utente.]nome_tabella_ambito... ) ]
  | MODIFY ( [colonna [tipodato] [DEFAULT espr]
             [vincolo_colonna] ...] ...)
  | [clausola_attributi_fisici]
  | {LOB_clausola_registr [, clausola_registr_LOB...]}
  | {MODIFY_LOB_clausola_registr [,,
               MODIFY_LOB_clausola_registr]}
  | {NESTED_TABLE_clausola_registr [,,
               NESTED_TABLE_clausola_registr...]}
  | DROP clausola_elimina
  | ALLOCATE EXTENT [ ( {SIZE intero [K | M]
                         | DATAFILE 'nomefile'
                         | INSTANCE intero} ...) ]
  | DEALLOCATE UNUSED [KEEP intero [K | M] ] } ...
  | { [ENABLE {clausola_abilita | TABLE LOCK}
      | DISABLE {clausola_disabilita | TABLE LOCK} ] ...
      | [PARALLEL clausola_parallelia] {NOCACHE | CACHE} ]
  | RENAME TO nome_nuova_tabella
  | OVERFLOW {clausola_attributi_fisici | INCLUDING nome_colonna
              | ALLOCATE EXTENT [ ( {SIZE intero [K | M]
                         | DATAFILE 'nomefile'
                         | INSTANCE intero} ...) ]
              | DEALLOCATE UNUSED [KEEP intero [K | M] ] } ...
  | ADD OVERFLOW [ {clausola_attributi_fisici
                  | PCTTHRESHOLD intero
                  | INCLUDING nome_colonna
                  | TABLESPACE tablespace} ...]
  | MODIFY PARTITION nome_partizione { clausola_attributi_fisici
                                         | [LOGGING | NOLOGGING] } ...
  | MOVE PARTITION nome_partizione { clausola_attributi_fisici
                                         | [LOGGING | NOLOGGING]
                                         | TABLESPACE tablespace
                                         | PARALLEL clausola_parallelia } ...
}
```

```

| ADD PARTITION [nome_nuova_partizione]
|   VALUES LESS THAN (elenco_valori) {
|     clausola_attributi_fisici
|     | [LOGGING | NOLOGGING]
|     | TABLESPACE tablespace } ...
|
| DROP PARTITION nome_partizione
| TRUNCATE PARTITION nome_partizione [DROP STORAGE | REUSE
|                                     STORAGE]
|
| SPLIT PARTITION nome_vecchia_partizione AT (elenco_valori)
|   [INTO ( PARTITION [div_partizione_1]
|             [clausola_attributi_fisici
|             | [LOGGING | NOLOGGING]
|             | TABLESPACE tablespace ] ...
|             , PARTITION [div_partizione_2]
|               [clausola_attributi_fisici
|               | [LOGGING | NOLOGGING]
|               | TABLESPACE tablespace ] ...) ]
|             [ PARALLEL clausola_parallelia] ...
|
| EXCHANGE PARTITION nome_partizione
|   WITH TABLE nome_tabella_non_divisa
|     [{INCLUDING | EXCLUDING} INDEXES]
|     [{WITH | WITHOUT} VALIDATION ]
|
|   MODIFY PARTITION UNUSABLE LOCAL INDEXES
|   MODIFY PARTITION REBUILD UNUSABLE LOCAL INDEXES }

clausola_attributi_fisici ::=

[ PCTFREE intero
| PCTUSED intero
| INITTRANS intero
| MAXTRANS intero
| STORAGE registra]

LOB_clausola_registra ::=

LOB (lob_elem [, lob_elem ...]) STORE AS
  [lob_nomeseg]
  [( TABLESPACE tablespace
    | STORAGE registra_
    | CHUNK intero
    | PCTVERSION intero
    | CACHE
    | NOCACHE LOGGING | NOCACHE NOLOGGING
    | INDEX [lob_nome_indice]
    [( TABLESPACE tablespace
      | STORAGE registra
      | INITTRANS intero
      | MAXTRANS intero ) ] )]

MODIFY_LOB_clausola_registra::=

MODIFY LOB (lob_elem)
( STORAGE registra
  | PCTVERSION intero
  | CACHE

```

```

| NOCACHE LOGGING | NOCACHE NOLOGGING
| ALLOCATE EXTENT [ ( {SIZE intero [K | M]
| | DATAFILE 'nomefile'}
| | INSTANCE intero} ) ]
| DEALLOCATE UNUSED [KEEP intero [K | M] ]
| INDEX [lob nome indice]
| ( STORAGE registra
| | INITTRANS intero
| | MAXTRANS intero
| | ALLOCATE EXTENT [ ( {SIZE intero [K | M]
| | | DATAFILE 'nomefile'}
| | | INSTANCE intero} ) ]
| | DEALLOCATE UNUSED [KEEP intero [K | M] ] ) ]
NESTED TABLE clausola_registra ::=  

NESTED TABLE elem_annidato STORE AS tabella_registra

```

DESCRIZIONE 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.

Incrementando la lunghezza di una colonna NOT NULL, senza specificare NULL, questa viene lasciata invariata a NOT NULL.

Le viste che fanno riferimento a una tabella tramite select * from non funzionano dopo che una colonna è stata aggiunta alla tabella a meno che non vengano eliminate e ricreate.

ALLOCATE EXTENT consente di allocare in modo esplicito una nuova estensione. ENABLE e DISABLE abilitano e disabilitano le limitazioni. Tutte le altre funzionalità di ALTER TABLE funzionano allo stesso modo che nel comando CREATE TABLE, fatta eccezione 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.

Quando vengono letti dei blocchi dal database, ORACLE li memorizza nella SGA. ORACLE gestisce un elenco (LRU) dei blocchi utilizzati più di recente; quando viene richiesto spazio aggiuntivo all'interno della SGA, i blocchi di dati utilizzati meno di recente vengono rimossi dalla SGA.

CACHE modifica questo funzionamento specificando che i blocchi per la tabella in questione devono essere letti e inseriti all'estremità “dei blocchi utilizzati più di recente” dell'elenco LRU. Quindi, i blocchi rimangono nella SGA più a lungo. Questa opzione è utile se le tabelle sono di piccole dimensioni e abbastanza statiche. NOCACHE (l'opzione di default) 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 Parallel Query). **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 un server parallelo per l'elaborazione in parallelo delle query. Un intero *n* specifica il numero delle istanze disponibili su cui è possibile suddividere la tabella.

È possibile utilizzare **ALTER TABLE** per eseguire sulle partizioni i comandi **ADD**, **DROP**, **EXCHANGE**, **MODIFY**, **MOVE**, **SPLIT** o **TRUNCATE**. Vedere il Capitolo 17 e **CREATE TABLE**.

Le clausole di registrazione LOB specificano l'area di memorizzazione da utilizzare per registrazioni fuori linea di dati LOB memorizzati internamente. Vedere il Capitolo 27 e il comando **STORAGE**.

È inoltre possibile modificare le registrazioni di tabelle annidate (per le relative registrazioni fuori linea).

Vedere il Capitolo 26 per ulteriori informazioni sulle tabelle annidate.

ALTER TABLESPACE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE **CREATE TABLESPACE**, **DROP TABLESPACE**, **STORAGE**, Capitolo 19

SINTASSI **ALTER TABLESPACE** *tablespace*

```
{ ADD DATAFILE definizione_file [,definizione_file] |
  [AUTOEXTEND definizione_file [, definizione_file] [ON | OFF]
    [NEXT intero[K | M]]
    [MAXSIZ [UNLIMITED | intero [K | M]]]
  RENAME DATAFILE file [,file]... TO file [,file]...
  DEFAULT STORAGE registra [MINIMUM EXTENT intero [K | M] |
    [READ WRITE | READ ONLY] |
    ONLINE | OFFLINE [NORMAL | TEMPORARY | IMMEDIATE] |
    [BEGIN BACKUP| END BACKUP]
    [PERMANENT | TEMPORARY ]
    [COALESCE] }
```

DESCRIZIONE *tablespace* è il nome dello spazio occupato da una tabella. **ADD DATAFILE** aggiunge al tablespace un file o una serie di file descritti in accordo con *definizione_file*, che specifica i nomi e le dimensioni dei file del database:

'*file*' [SIZE *intero* [K | M]] [REUSE]

Il formato del nome del file è specifico per il sistema operativo. **SIZE** rappresenta il numero di byte riservati per questo file. Specificando il suffisso K, questo valore viene moltiplicato per 1024, invece con M viene moltiplicato per 1048576. **REUSE** (senza **SIZE**) distrugge i contenuti di qualsiasi file con il nome specificato e assegna questo nome al database. **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.

Si può 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**).

RENAME cambia il nome del file contenente un tablespace esistente. Il tablespace deve essere sconnesso al momento in cui avviene il cambio del nome. Si noti che **RENAME**, in realtà, non rinomina i file; associa solamente i loro nuovi nomi con il tablespace in questione. La modifica vera e propria dei nomi dei file all'interno del sistema operativo deve essere effettuata all'interno del sistema operativo stesso. Per cambiare in modo appropriato i nomi ai file, per prima cosa si esegua il comando **ALTER TABLESPACE OFFLINE**, si rinominino i file all'interno del sistema operativo, eseguendo un **RENAME** dei file con **ALTER TABLESPACE** e, successivamente, si esegua il comando **ALTER TABLESPACE ONLINE**.

DEFAULT STORAGE definisce lo spazio di memorizzazione di default per tutti i futuri oggetti creati nel tablespace in questione, fatta eccezione per quei valori di default che vengono reimpostati, ad esempio tramite un'operazione di **CREATE TABLE**. **ONLINE** riporta in linea il tablespace. **OFFLINE** lo sconnette, senza aspettare che i suoi utenti si scolleghino (opzione **IMMEDIATE**) oppure dopo che tutti gli utenti hanno terminato la sessione (opzione **NORMAL**).

MINIMUM EXTENT imposta la dimensione minima per qualsiasi estensione creata nel tablespace. **COALESCE** combina le estensioni libere vicine tra loro nel tablespace in un numero minore di estensioni libere più grandi.

I tablespace **READ ONLY** non vengono mai aggiornati da ORACLE e possono essere registrati su supporti di sola lettura. I tablespace di sola lettura non vengono aggiornati. Tutti i tablespace vengono creati con la possibilità di lettura e scrittura. Per riportare un tablespace di sola lettura nello stato lettura-scrittura, si utilizza la clausola **READ WRITE**.

BEGIN BACKUP può essere eseguito in qualsiasi momento. Esso assicura che tutti i file del database presenti nel tablespace vengano salvati nel corso del successivo salvataggio di sistema. **END BACKUP** indica che il salvataggio del sistema è finito. Se il tablespace è in linea, qualsiasi salvataggio del sistema deve salvare anche i redo log dell'archivio. Se, al contrario, non è in linea, questa operazione non è necessaria.

I 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. Un tablespace **PERMANENT** può contenere qualsiasi tipo di segmento di ORACLE.

ALTER TRIGGER

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE **CREATE TRIGGER**, **DISABLE**, **DROP TRIGGER**, **ENABLE**, **TRIGGER**, Capitolo 23

SINTASSI **ALTER TRIGGER** [*utente*.]*trigger* {**ENABLE** | **DISABLE** | **COMPILE** [**DEBUG**]}

DESCRIZIONE **ALTER TRIGGER** abilita, disabilita o ricompila un trigger PL/SQL. Vedere **CREATE TRIGGER** e il Capitolo 23 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**. Poiché i trigger dipendono da altri elementi, possono non risultare più 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

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE TYPE, CREATE TYPE BODY, Capitoli 4 e 25

SINTASSI ALTER TYPE [*utente.*]*nome_tipo*
{ COMPILE [SPECIFICATION | BODY] | REPLACE }
(*nome_attributo tipodato*[, *nome_attributo tipodato*]...
| [{MAP | ORDER} MEMBER *specifica_funzione*]
| [MEMBER {*specifica_procedura* | *specifica_funzione*}
[, MEMBER {*specifica_procedura* | *specifica_funzione*}]...]]
| [PRAGMA RESTRICT REFERENCES (*nome_metodo*, *limitazioni*)
[, PRAGMA RESTRICT REFERENCES (*nome_metodo*, *limitazioni*)]...])

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 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). Vedere il Capitolo 25 per alcuni esempi sui metodi e i corpi dei tipi.

ALTER USER

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE TABLESPACE, GRANT, CREATE PROFILE, CREATE USER, Capitolo 18

SINTASSI ALTER USER *utente*
{ IDENTIFIED {BY *password* | EXTERNALLY
| GLOBALLY AS 'CN=*utente*'}
| 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 ROLE { *ruolo* [, *ruolo*] ...
| ALL [EXCEPT *ruolo* [, *ruolo*] ...]
| NONE} } ...

DESCRIZIONE Si utilizza ALTER USER per modificare la password di un utente, il tablespace di DEFAULT (per gli oggetti di proprietà dell'utente) o i tablespace TEMPORARY (per segmenti temporanei utilizzati dall'utente). Senza ALTER USER, i valori di default per entrambi vengono impostati secondo i valori di default del primo tablespace (sia per gli oggetti che i segmenti temporanei) concesso all'utente come risorsa tramite GRANT. ALTER USER consente inoltre di modificare la quota, il profilo della risorsa o il ruolo di default (vedere CREATE USER). Si può utilizzare la clausola PASSWORD EXPIRE per forzare l'estinzione della password di un utente;

nel qual caso, l'utente deve 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.

ALTER VIEW

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE VIEW, DROP VIEW, Capitolo 17

SINTASSI ALTER VIEW [*utente.*]*vista* COMPILE

DESCRIZIONE ALTER VIEW ricompila una vista. Lo si può utilizzare per scoprire eventuali errori prima di eseguire la vista. Risulta particolarmente utile dopo aver modificato in qualche modo una tabella di base.

Per modificare una vista è necessario esserne il proprietario oppure disporre del privilegio di sistema ALTER ANY TABLE.

ALTRE 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 questo guida, assieme alla sintassi e a esempi d'uso.

DUMP(*stringa* [,*formato* [,*inizio* [,*lunghezza*]]])

DUMP visualizza il valore di *stringa* in un formato interno, in ASCII, ottale, decimale, esadecimale o in formato carattere.

NVL(*valore*, *sostituto*)

Se *valore* è NULL, la funzione NVL restituisce *sostituto*. NVL può essere utilizzata anche come funzione PL/SQL.

VSIZE(*espressione*)

VSIZE ritorna il numero di byte di cui ORACLE necessita per l'archiviazione dell'*espressione* nel database.

ANALYZE

TIPO Comando SQL

PRODOTTI Tutti

SINTASSI ANALYZE {INDEX | TABLE | CLUSTER}
 [*utente.*] {indice [PARTITION (*nome_partizione*)]
 | tabella [PARTITION (*nome_partizione*)]
 | cluster}
 { COMPUTE STATISTICS [FOR *clausola_for*]
 | ESTIMATE STATISTICS [FOR *clausola_for*]
 [SAMPLE *intero* {ROWS | PERCENT}]
 | DELETE STATISTICS
 | VALIDATE REF UPDATE
 | VALIDATE STRUCTURE [CASCADE]
 | LIST CHAINED ROWS [INTO [*utente.*] *tabella*] }

DESCRIZIONE 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 accurata quanto quella elaborata dal comando COMPUTE che fornisce le informazioni statistiche dell'ottimizzatore. Per controllare se un oggetto presenta una possibile corruzione dei dati, si può utilizzare la clausola VALIDATE STRUCTURE. Per analizzare una tabella, è necessario esserne il proprietario o possedere il privilegio di sistema ANALYZE ANY.

A COMPUTE STATISTICS ed ESTIMATE STATISTICS può essere associata la clausola FOR. Il formato della *clausola_for* è:

```
{FOR TABLE |  
  FOR ALL [INDEXED] COLUMNS [SIZE intero] |  
  FOR COLUMNS [SIZE intero] colonna [SIZE intero]... |  
  FOR ALL [LOCAL] INDEXES}
```

La clausola FOR consente di modificare il funzionamento di default di ANALYZE. Si può 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 che si basa sui costi, durante le operazioni di regolazione di tipo avanzato.

La clausola LIST CHAINED ROWS registra informazioni relative alle righe concatenate in una tabella specificata. Vedere il Capitolo 32 per ulteriori informazioni sulle tabelle CHAINED_ROWS.

ANALYZE può analizzare un solo oggetto alla volta. Per analizzare un intero schema, si può utilizzare la procedura DBMS.Utility.ANALYZE_SCHEMA; questa procedura richiede due parametri: il nome dello schema e il metodo da utilizzare (COMPUTE o ESTIMATE). Vedere *ORACLE Server Application Developer's Guide*.

AND

Vedere OPERATORI DI RICERCA TESTUALE, OPERATORI LOGICI, PRECEDENZE.

ANNIDAMENTO

L'annidamento è l'atto di inglobare un'istruzione, un'opzione, una query e così via all'interno di un'altra istruzione, opzione, query e così via.

ANSI

L'American National Standards Institute ha stabilito una serie di standard per il linguaggio SQL e per molti dei suoi elementi.

ANY

TIPO Operatore logico SQL

PRODOTTI Tutti

VEDERE ANCHE ALL, BETWEEN, EXISTS, IN, OPERATORI LOGICI, Capitolo 11

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 sottoquery, in cui ciascuna riga della sottoquery 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)
```

RESTRIZIONI *elenco* non può essere una serie di colonne di una sottoquery 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 una certa forma 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 !=. |

ESEMPI Si consideri la seguente tabella:

```
select * from GIORNALE
order by Pag, Sezione;
```

| ARGOMENTO | S | PAG |
|--------------|---|-----|
| Notizie | A | 1 |
| Vita moderna | B | 1 |
| Sport | D | 1 |
| Economia | E | 1 |
| Bridge | B | 2 |
| Meteo | C | 2 |
| Film | B | 4 |

| | | |
|-------------|---|----|
| Fumetti | C | 4 |
| Necrologi | F | 6 |
| Salute | F | 6 |
| Televisione | B | 7 |
| Nascite | F | 7 |
| Annunci | F | 8 |
| Editoriali | A | 12 |

La logica di ANY, utilizzato sulla tabella precedente, viene mostrata nelle query che seguono. Si tenga presente che l'elenco dei valori (4,2,7) potrebbe anche essere semplicemente una sottoquery. Si osservino con attenzione i numeri delle pagine che soddisfano la query in ciascuno dei casi seguenti:

```
select * from GIORNALE
where Pag = ANY (4,2,7);
```

| ARGOMENTO | S | PAG |
|-------------|---|-----|
| Meteo | C | 2 |
| Televisione | B | 7 |
| Nascite | F | 7 |
| Fumetti | C | 4 |
| Film | B | 4 |
| Bridge | B | 2 |

```
select * from GIORNALE
where Pag > ANY (4, 2, 7);
```

| ARGOMENTO | S | PAG |
|-------------|---|-----|
| Editoriali | A | 12 |
| Televisione | B | 7 |
| Nascite | F | 7 |
| Annunci | F | 8 |
| Fumetti | C | 4 |
| Film | B | 4 |
| Necrologi | F | 6 |
| Salute | F | 6 |

```
select * from GIORNALE
where Pag >= ANY (4,2,7);
```

| ARGOMENTO | S | PAG |
|-------------|---|-----|
| 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 |

select * from GIORNALE
where Pag < ANY (4,2,7);

| ARGOMENTO | S | PAG |
|--------------|---|-----|
| Notizie | A | 1 |
| Sport | D | 1 |
| Economia | E | 1 |
| Meteo | C | 2 |
| Vita moderna | B | 1 |
| Fumetti | C | 4 |
| Film | B | 4 |
| Bridge | B | 2 |
| Necrologi | F | 6 |
| Salute | F | 6 |

select * from GIORNALE
where Pag <= ANY (4,2,7);

| ARGOMENTO | S | PAG |
|--------------|---|-----|
| 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 |

select * from GIORNALE
where Pag != ANY (4,2,7);

| ARGOMENTO | S | PAG |
|------------|---|-----|
| 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 |

APPEND

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE CHANGE, DEL, EDIT, LIST, Capitolo 5

SINTASSI A[PPEND] *testo*

DESCRIZIONE APPEND è una funzionalità dell'editor a riga di comando di SQL*PLUS. Questo comando posiziona il testo alla fine della riga corrente nel buffer corrente, senza interporre spazi. Se si vuole 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 vuole 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. Ad esempio, vi sono applicazioni che servono come sistemi per l'acquisizione di ordini di merci.

ARCHIVE_LOG

TIPO Comando SQL

PRODOTTI Tutti

SINTASSI ARCHIVE_LOG [THREAD *intero*]
 {{SEQ *intero* |
 CHANGE *intero* |
 CURRENT |
 GROUP *intero* |
 LOGFILE *file* |
 NEXT |
 ALL |
 START} [TO *locazione*] } |
 STOP}

DESCRIZIONE ARCHIVE_LOG fa parte del comando ALTER SYSTEM. Consente di controllare l'archiviazione dei gruppi di redo log all'interno di un thread specifico.

SEQ identifica il gruppo da archiviare manualmente nel thread. CHANGE archivia manualmente un gruppo specificato dal numero di modifica del sistema ed effettua un cambio del log, se l'archiviazione riguarda il gruppo di file redo log corrente. CURRENT archivia manualmente il gruppo corrente, forzando un cambio

del file di log. GROUP archivia manualmente un gruppo specifico. LOGFILE archivia manualmente un gruppo specificato dal relativo nome di file di log. NEXT archivia manualmente il successivo gruppo di file. ALL archivia manualmente tutti i gruppi di file di log nei thread che risultano pieni, ma non sono stati archiviati.

START abilita l'archiviazione automatica di gruppi di file di log. **STOP** disabilita l'archiviazione automatica.

TO specifica una collocazione per i dati da archiviare. Quest'ultima deve essere specificata in modo completo.

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 media.

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.

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, che fornisce un valore che la funzione può utilizzare.

ARRAY, ELABORAZIONE

L'elaborazione di un array è eseguita su insiemi di dati anziché su una riga per volta. In alcune utilità 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 diversi tipi di dato. Il numero massimo di dati di un array variabile viene impostato alla sua creazione (con CREATE TYPE). Vedere il Capitolo 26 per degli esempi dettagliati riguardanti la creazione e il recupero di dati su array variabili e tabelle annidate (l'altro tipo di collettori).

ARRAYSIZE (SQL*PLUS)

Vedere SET.

AS

TIPO Delimitatore SQL

PRODOTTI Tutti

VEDERE ANCHE ALIAS, TO_CHAR, Capitoli 6 e 8

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

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; CHR

SINTASSI ASCII(*stringa*)

DESCRIZIONE ASCII è l'acronimo di “American Standard Code for Information Interchange”, una convenzione che consente di utilizzare i 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 254. Quelli tra 0 e 127 hanno uno standard ben definito. Quelli superiori al 127 (“insieme ASCII esteso”) differiscono da paese a paese, da applicazione ad applicazione e a seconda del costruttore del computer. La lettera A, ad 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 Mezzanotte, ASCII(Mezzanotte) from COMFORT;

```
MEZZANOTTE ASCII(MEZzanotte)
```

```
-----
```

| | |
|------|----|
| 42.3 | 52 |
| 71.9 | 55 |
| 61.5 | 54 |
| 39.8 | 51 |
| -1.2 | 45 |
| 66.7 | 54 |
| 82.6 | 56 |
| -1.2 | 45 |

```
select ASCII('.'), ASCII(.5),
       ASCII('M'), ASCII('MULLER')
from DUAL;
ASCII('..') ASCII(.5) ASCII('M') ASCII('MULLER')
```

```
-----
```

| | | | |
|----|----|----|----|
| 46 | 46 | 77 | 77 |
|----|----|----|----|

ASIN

TIPO Comando SQL

PRODOTTI Tutti

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.

Astratti, tipi di dati

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. Ad esempio, un tipo di dato astratto per contenere indirizzi può essere rappresentato dalle colonne seguenti:

| | |
|-------|--------------|
| Via | VARCHAR2(50) |
| Città | VARCHAR2(25) |
| Prov | CHAR(2) |
| CAP | NUMBER |

I tipi di dati astratti possono avere dei metodi associati. Vedere il Capitolo 4 e il Capitolo 25 per un'introduzione ai tipi di dati astratti.

ATAN

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ACOS, ASIN, ATAN2, COS, COSH, EXP, LN, LOG, SIN, SINH, TAN, TANH

SINTASSI ATAN(*valore*)

DESCRIZIONE ATAN restituisce l'arcotangente di un valore. I valori di input sono infiniti; gli output sono espressi in radianti.

ATAN2

TIPO Comando SQL

PRODOTTI Tutti

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.

ATTRIBUTO

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.

AUDIT (forma 1, oggetti di schema)

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE DIZIONARIO DI DATI, NOAUDIT, PRIVILEGIO

SINTASSI AUDIT { *opzione, opzione, ... | ALL* }

ON {[*utente*.]*oggetto* | DEFAULT}

[BY SESSION | ACCESS]

[WHENEVER [NOT] SUCCESSFUL]

DESCRIZIONE *opzione* si riferisce alle opzioni descritte successivamente. *utente* rappresenta il nome del proprietario dell'oggetto. *oggetto* può essere una tabella, una vista o un sinonimo. Per controllare uno di questi elementi, è necessario esserne il proprietario oppure possedere il privilegio di sistema AUDIT ANY. *opzione* specifica quali comandi debbano essere controllati per un oggetto. Le opzioni sono: ALTER, AUDIT, COMMENT, DELETE, EXECUTE, GRANT, INDEX, INSERT, LOCK, READ, RENAME, SELECT e UPDATE. GRANT controlla sia il comando GRANT che REVOKE. ALL li controlla tutti.

ON *oggetto* specifica il nome dell'oggetto da controllare e include una tabella, una vista, il sinonimo di una tabella o una vista, una procedura, una funzione memorizzata, un package o uno snapshot. ON DEFAULT significa una variazione alle opzioni della tabella DEFAULT, che controlla le opzioni di AUDIT assegnate a tutte le tabelle e richiede l'autorità di DBA. Questo valore di default per tutte le tabelle viene impostato utilizzando la parola DEFAULT anziché il nome di una tabella.

```
audit grant, insert, update, delete on default;
```

ALTER, EXECUTE e INDEX non possono essere utilizzati per le viste. Le opzioni di default per una vista vengono create dall'unione delle opzioni di ciascuna delle tabelle sottostanti più le opzioni di DEFAULT. I controlli delle tabelle e delle viste sono indipendenti.

Per i sinonimi, le opzioni sono le stesse delle tabelle.

Per le sequenze, le opzioni sono ALTER, AUDIT, GRANT e SELECT.

EXECUTE può essere utilizzato solamente per procedure, funzioni, tipi di dati astratti, librerie o package, per i quali è anche possibile effettuare l'AUDIT, il GRANT, e il RENAME di opzioni.

Per gli snapshot, è possibile effettuare solo l'audit di SELECT e per le directory solo quello di READ.

BY ACCESS o BY SESSION scrive una riga in una tabella di auditing per ciascun comando o ciascuna sessione dell'utente, utilizzando la tabella su cui si sta effettuando il controllo. Se questa clausola viene omessa, l'opzione di default è BY SESSION. BY ACCESS esegue anche BY SESSION.

NOAUDIT inverte l'effetto di AUDIT.

Se l'accesso alle tabelle remote è realizzato tramite un collegamento al database, qualsiasi controllo sul sistema remoto si basa sulle opzioni impostate all'interno di quest'ultimo. Le informazioni di controllo vengono scritte in una tabella di nome SYS.AUD\$; vedere la voce VISTE DEL DIZIONARIO DI DATI e il Capitolo 24 per informazioni relative alle viste di questa tabella.

ESEMPI Ciascun comando di auditing seguente cerca di utilizzare update o delete sulla tabella LAVORATORE:

```
audit update, delete on LAVORATORE by access;
```

I comandi di auditing seguenti non hanno avuto successo nell'accesso a LAVORATORE:

```
audit all on LAVORATORE whenever not successful;
```

AUDIT (forma 2, istruzioni SQL)

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE DATABASE LINK, DIZIONARIO DI DATI, NO-AUDIT, PRIVILEGIO

SINTASSI AUDIT {*opzione istruzione | privilegio di sistema*}
 [,{*opzione istruzione | privilegio di sistema*}][...]
 [BY *utente[,utente]...*]
 [BY SESSION | ACCESS]
 [WHENEVER [NOT] SUCCESSFUL]

DESCRIZIONE Il secondo formato del comando AUDIT controlla l'utilizzo di una facilità di sistema e richiede, per essere eseguito, di disporre del privilegio di sistema AUDIT SYSTEM.

Le *opzioni istruzione* sono le seguenti.

| OPZIONE | FUNZIONE DI AUDITING |
|----------------------|--|
| CLUSTER | Crea, modifica, elimina o tronca un cluster. |
| DATABASE LINK | Crea o elimina un collegamento a database. |
| DIRECTORY | Crea o elimina una directory. |
| EXISTS | Istruzioni SQL che falliscono perché l'oggetto esiste già (Trusted Oracle). |
| INDEX | Crea, modifica o elimina un indice. |
| NOT EXISTS | Istruzioni SQL che falliscono perché l'oggetto non esiste. |
| PROCEDURE | Crea, modifica o elimina una procedura, una funzione o un package e 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 no l'audit di istruzioni SQL. |
| SYSTEM GRANT | Esegue o no l'audit di privilegi di sistema e ruoli. |
| TABLE | Crea, modifica, elimina o tronca una tabella. |
| TABLESPACE | Crea, modifica o elimina un 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. |

È inoltre possibile controllare ogni singolo comando compreso nell'opzione dell'istruzione. ORACLE fornisce il seguente gruppo di opzioni per istruzione:

- CONNECT controlla i collegamenti e le sconnessioni da ORACLE;
- DBA controlla i comandi che richiedono l'autorità di DBA, quali GRANT, REVOKE, AUDIT, NOAUDIT, CREATE o ALTER TABLESPACE e CREATE o DROP PUBLIC SYNONYM;
- RESOURCE controlla le operazioni di CREATE e DROP di tabelle, cluster, viste, indici, tablespace e sinonimi;
- ALL controlla tutte queste facilità;
- BY *utente* controlla solo le istruzioni SQL eseguite da utenti particolari. Il funzionamento di default è di controllare tutti gli utenti;
- WHENEVER [NOT] SUCCESSFUL scrive una riga in una tabella di auditing solo nel caso che un tentativo di accesso ad una tabella controllata o a una facilità di sistema abbia (o no, NOT) successo; se questa clausola viene omessa, viene scritta una riga, che l'accesso abbia successo oppure no;
- NOAUDIT inverte gli effetti di AUDIT.

Entrambi i formati eseguono il commit di qualsiasi modifica pendente nei confronti del database. Se l'accesso alle tabelle remote avviene tramite un collegamento a database, qualsiasi controllo sul sistema remoto è basato su opzioni impostate all'interno di quest'ultimo. Le informazioni dell'audit vengono scritte in una tabella di nome SYS.AUDIT_TRAIL; vedere VISTE DEL DIZIONARIO DI DATI e il Capitolo 32 per informazioni relative alle viste di questa tabella.

ESEMPIO Questo comando rivela tutti i collegamenti che hanno avuto successo:

```
audit CONNECT whenever successful;
```

AUDITING

Le operazioni di auditing rappresentano un insieme di software e funzionalità per il dizionario di dati di ORACLE che consentono al DBA e agli utenti di seguire le tracce del funzionamento del database. Il DBA può impostare un'attività di auditing di default. Le informazioni dell'auditing vengono memorizzate nel dizionario dei dati e le istruzioni SQL controllano quale informazione viene memorizzata. Ad esempio, è possibile ottenere una traccia di ciascun tentativo di aggiornamento dei dati in una tabella oppure solo dei tentativi che hanno avuto successo. Analogamente, si possono registrare tutti i collegamenti effettuati con ORACLE oppure soltanto i tentativi riusciti di eseguire operazioni riservate al DBA.

AUTOCOMMIT

AUTOCOMMIT è un comando di SQL*PLUS utilizzato 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.

AUTORIZZAZIONE

Vedere PRIVILEGIO.

AVG

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE COMPUTE, GROUP FUNCTIONS, Capitolo 7

SINTASSI AVG([DISTINCT] *valore*)

DESCRIZIONE AVG è la media dei valori contenuti in un gruppo di righe. DISTINCT forza l'esecuzione 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 aprire un database e di renderlo disponibile.

B-TREE

B-TREE rappresenta una struttura indicizzata utilizzata da ORACLE per creare e memorizzare gli indici.

BACKGROUND, PROCESSO

Un processo in background è uno dei processi utilizzati da un'istanza di ORACLE nel corso dell'esecuzione contemporanea di più processi, allo scopo di eseguire e coordinare operazioni nell'interesse degli utenti simultaneamente collegati a un database. I processi di base si chiamano ARCH (archiviazione), DBWR (scrittura nel database), LGWR (scrittura del log), PMON (controllo del processo) e SMON (controllo del sistema) ed esistono per tutto il tempo in cui è attiva un'istanza.

BACKUP IN LINEA

Il backup in linea è la funzione di ORACLE che archivia i dati durante l'esecuzione del database. Non è necessario chiudere il DBA per archiviare i dati di un database. Anche i dati in corso di accesso possono essere archiviati.

BEGIN

TIPO Istruzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE BLOCCO, STRUTTURA; DECLARE; END; EXCEPTION; TERMINATOR; Capitolo 22

SINTASSI [<

[DECLARE]

BEGIN

... logica del blocco ...

END [blocco];

DESCRIZIONE BEGIN è l'istruzione d'apertura della sezione esecutiva di un blocco PL/SQL. Può essere seguita da qualsiasi logica di PL/SQL valida e da un gestore di eccezioni e viene chiusa dall'istruzione END. Vedere BLOCCO, STRUTTURA e il Capitolo 22 per ulteriori dettagli.

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 deve 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* e il Capitolo 27.

BINDING, FASE

La fase di binding è la fase dell'elaborazione di un'istruzione SQL durante la quale tutte le variabili vengono rese note all'RDBMS in modo che i valori reali possano essere utilizzati nell'esecuzione dell'istruzione.

BINDING, VARIABILE

Una variabile binding è una variabile contenuta in un'istruzione SQL che deve essere sostituita con un valore valido o l'indirizzo di un valore in modo che l'istruzione posse essere eseguita correttamente.

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 di utilizzare un indice bitmap per quella colonna. *Vedere* il Capitolo 19 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, memorizzato all'interno del database. *Vedere DATI, TIPI* e il Capitolo 27.

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 sui dati del database. Ne esistono di tre tipi: i blocchi per le operazioni del dizionario, i blocchi per le definizioni del dizionario e i blocchi sulle definizioni delle tabelle.

BLOCCO

Unità di base di memorizzazione (fisica e logica) per tutti i dati di ORACLE. Il numero di blocchi allocati per tabella di ORACLE dipende dal tablespace in cui la tabella viene creata. La dimensione del blocco di ORACLE varia a seconda del sistema operativo e può differire dalla dimensione del blocco del sistema operativo dell'host. Le comuni dimensioni dei blocchi sono 512 byte (caratteri) e 2048 byte. Per conoscere la dimensione di un blocco sul proprio sistema operativo, si faccia riferimento alla *Installation and User's Guide*. Lo stesso vale anche per "pagina" di memoria.

BLOCCO (LOCK)

Un blocco è una restrizione temporanea dell'accesso degli altri utenti. I tipi di blocco attivabili sono SHARE, SHARE UPDATE, EXCLUSIVE, SHARE EXCLUSIVE, ROW SHARE e ROW EXCLUSIVE.

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 22

DESCRIZIONE I blocchi PL/SQL possono ora essere racchiusi in SQL*PLUS o in un qualsiasi altro linguaggio di programmazione, tramite l'utilizzo dei precompilatori di ORACLE. I blocchi PL/SQL risultano strutturati nel modo seguente:

```
[<<blocco>>]
[DECLARE
  .. dichiarazioni (CURSOR, VARIABLE ed EXCEPTION)...]

BEGIN
  ...
  ... logica blocco (codice eseguibile)...

[EXCEPTION
  ...
  ... logica del gestore di eccezione (per errori fatali)...]

END [blocco];
```

Come risulta dalle parentesi quadre, sia la sezione DECLARE che EXCEPTION sono opzionali. Un blocco può opzionalmente essere etichettato tramite un nome, che deve 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, che 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 riguardo a ciò).

Se viene fatto riferimento a una variabile nella dichiarazione di un cursore, occorre che sia specificata prima della dichiarazione del cursore. *Vedere* il Capitolo 22 per degli esempi riguardanti le strutture dei blocchi e i cicli.

```
ESEMPIO DECLARE
    pi      constant NUMBER(9,7) := 3.1415926;
    area   NUMBER(14,2);
    cursor rad_cursore is
        select * from RAGGIO_VAL;
    rad_val rad_cursore%ROWTYPE;
BEGIN
    open rad_cursore;
    loop
        fetch rad_cursore into rad_val;
        exit when rad_cursore%NOTFOUND;
        area := pi*power(rad_val.raggio,2);
        insert into AREE values (rad_val.raggio, area);
    end loop;
    close rad_cursore;
END;
.
```

BLOCCO A LIVELLO DI RIGA

Si tratta di un meccanismo di blocco per cui gli aggiornamenti ai dati vengono effettuati bloccando solo le righe intaccate (e non l'intera pagina).

BLOCCO CONDIVISO

Si tratta di un blocco che permette agli altri utenti di leggere i dati ma non di modificarli.

BLOCCO DELLA MODALITÀ ESCLUSIVA

Vedere MODALITÀ ESCLUSIVA.

BLOCCO DI INTESTAZIONE DI SEGMENTO

Si tratta del primo blocco della prima parte di un segmento e contiene, tra le altre cose, un elenco delle estensioni del segmento stesso.

BLOCCO DI RECORD

Il blocco (lock) di record evita che due utenti aggiornino 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 effettuare 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; nello stesso istante vari utenti possono porre diverse limitazioni sugli stessi dati.

BLOCCO FISICO

Un blocco fisico è una locazione di memoria di massa, in genere contenente 512 byte. Non va confuso con i blocchi di ORACLE; *vedere BLOCCO*. Computer diversi hanno blocchi fisici di dimensioni differenti.

Si tratta di un meccanismo di protezione che permette agli utenti di leggere i dati ma impedisce loro di modificarli.

BLOCCO IN MODALITÀ CONDIVISA

È in pratica un sinonimo di blocco a livello di riga, un metodo di protezione per cui durante gli aggiornamenti vengono bloccate le sole righe interessate e non l'intera tabella.

BLOCCO PER AGGIORNAMENTO CONDIVISO

Si tratta di un blocco che permette ad altri utenti di leggere e bloccare i dati.

BREAK

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE CLEAR, COMPUTE, Capitoli 5 e 13

SINTASSI BRE[AK] ON { REPORT | *espressione* | ROW | PAG[E] } ...
 [SKIP[P] *linee* | [SKIP] PAGE]
 [NODUP[LICATES] | DUP[LICATES]]
 BRE[AK]

DESCRIZIONE Un break ha luogo quando SQL*PLUS rileva un cambiamento specifico, quale la fine di una pagina o la variazione del valore di un'espressione. Esso 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, ad 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 break.

REPORT ON causa un break alla fine di un report o di una query. Una modifica nel valore di un'espressione può verificarsi in qualsiasi momento prima del break di REPORT o PAGE e vi possono essere, in un'unica istruzione BREAK, diverse clausole dell'espressione ON; tuttavia, il loro ordine all'interno del comando BREAK dovrebbe essere lo stesso di quello della clausola order by dell'istruzione select. Ciò significa che ciascuna espressione che compare nel comando BREAK deve essere inserita anche nella clausola order by dell'istruzione select, nella stessa sequenza oppure il risultato non avrà senso.

In più, l'ordine dovrebbe essere dal raggruppamento più grande al più piccolo (ad esempio, ON Società, ON Reparto, ON Progetto).

ON ROW causa un break per ogni riga selezionata.

ON PAGE causa un break alla fine di ciascuna pagina ed è indipendente dai break causati dall'espressione ON ROW oppure ON REPORT.

SKIP salta un numero di righe, mentre PAGE o SKIP PAGE salta a una nuova pagina, prima di stampare il risultato dell'operazione COMPUTE associata al break.

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 il break.

CLEAR BREAKS rimuove qualsiasi BREAK esistente.

Per esempi relativi a queste opzioni, vedere il Capitolo 13.

BTITLE (titolo a fondo pagina)

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE ACCEPT, DEFINE, PARAMETRI, REPFOOTER, REPHEADER, SETHEADSEP, TTITLE, Capitolo 13

SINTASSI BTI[TLE] [*opzione* ['*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 e *variabile* rappresenta una variabile definita dall'utente o una variabile gestita dal sistema, incluse SQL.LNO, il numero della riga 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.

COL[UMN] *n* salta direttamente alla posizione *n* (dal margine sinistro) della linea corrente.

S[KIP] *n* stampa *n* righe vuote. Se non viene specificato alcun *n*, viene stampata una riga vuota. Se *n* è uguale a 0, non viene stampata alcuna riga vuota e la posizione corrente per la stampa diventa la posizione 1 della riga corrente (quella più a sinistra nella pagina).

TAB *n* si sposta di *n* posizioni in avanti (o all'indietro se *n* è negativo).

LE[FT], CE[ENTER] e RI[GHT], rispettivamente, allineano a sinistra, centrano e allineano a destra i dati sulla riga corrente. Qualsiasi testo o variabile che segue questi comandi viene allineato come un gruppo, fino alla fine del comando oppure a sinistra (LEFT), al centro (CENTER), a destra (RIGHT) o in colonna (COLUMN). CENTER e RIGHT utilizzano il valore impostato dal comando SET LINESIZE per stabilire dove inserire il testo o la variabile.

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, viene annullato e riassegnato il precedente formato fino a quel momento attivo. 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 di default per SQL*PLUS.

I valori che si riferiscono a date vengono stampati in accordo al formato di default, a meno che non sia stata caricata una variabile con una data riformattata tramite TO_CHAR.

In un unico btitle può essere specificato qualsiasi numero di opzioni, testi e variabili. Ciascuno di questi elementi viene stampato nell'ordine specificato e ognuno viene posizionato e formattato nel modo specificato dalle clausole che lo precedono.

BUFFER

In termini generali, un *buffer* è una specie di blocco degli appunti, solitamente contenuto nella memoria del computer, in cui i comandi vengono preparati per la modifica e l'esecuzione.

In SQL*PLUS, un buffer rappresenta un'area di memoria per la modifica di comandi SQL e SQL*PLUS. 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 redo log sono spazi temporanei di memorizzazione di blocchi redo log del database utilizzati in quel momento dagli gli utenti.

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, che per i dati richiesti da ORACLE per il supporto agli utenti.

CACHE, GESTORE

Il gestore della cache è il processo che si accerta che tutte le modifiche effettuate dal software vengano riportate su disco nel giusto ordine.

CAMPO

In una tabella, un campo equivale all'informazione contenuta all'intersezione di una riga e di una colonna. Campo è generalmente sinonimo di colonna, sebbene spesso possa riferirsi ai valori in essa contenuti.

CARATTERI, FUNZIONI

TIPO Funzioni SQL

PRODOTTI Tutti

VEDERE ANCHE CONVERSIONE, FUNZIONI; NUMERI, FUNZIONI; ALTRE FUNZIONI; Capitolo 6

DESCRIZIONE Di seguito è riportato un elenco in ordine alfabetico di tutte le attuali funzioni per caratteri presenti in SQL di ORACLE. Ciascuna di esse è trattata a parte all'interno di questa guida sotto il proprio nome, con il suo proprio formato e modalità di utilizzo.

Nomi e modalità di utilizzo delle funzioni

|| lega o 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)`

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 ricorrenze della sequenza di caratteri specificata da *car*.

`NLS_INITCAP(stringa[, 'NLS_SORT=ordine'])`

NLS_INITCAP sta per National Language Support INITIAL 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.

`NLSORT(carattere)`

ORACLE utilizza il National Language Support SORT. Questa funzione fornisce il valore della sequenza di confronto di un dato carattere in base all'opzione National Language Support scelta per il sito.

`REPLACE(stringa, if [,then])`

REPLACE restituisce *stringa* con ciascuna occorrenza di *if* sostituita con *then* (zero o più caratteri). Se non viene specificata alcuna stringa *then*, tutte le occorrenze di *if* vengono rimosse. 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 ricorrenze 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. Si può utilizzare SOUNDEX per confrontare i nomi che possono avere piccole differenze di pronuncia, ma sono di fatto lo stesso nome.

SUBSTR(*stringa*, *inizio* [,*conta*])

SUB STRing ritaglia una parte di una stringa a partire dalla posizione *inizio* e contando un numero *conta* di caratteri a partire da *inizio*.

SUBSTRB(*stringa*, *inizio* [,*conta*])

SUB STRing Byte è analoga a SUBSTR eccetto che può gestire stringhe di più byte per il National Language Support.

TRANSLATE(*stringa*,*if*,*then*)

Questa funzione traduce una stringa, carattere per carattere, in base a una corrispondenza di posizione di caratteri nella stringa *if* con caratteri nella stringa *then*. Vedere REPLACE.

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 auditing. Le opzioni sono ENTRYID, SESSIONID e TERMINAL.

VSIZE(*stringa*)

VSIZE fornisce la dimensione di memorizzazione di *stringa* in ORACLE.

CARATTERI NON VISUALIZZABILI

I caratteri non visualizzabili sono stringhe di comando utilizzate per ottenere alcune funzioni video (come ad esempio la cancellazione dello schermo o lo spostamento di un cursore).

CATEGORIA

Una categoria viene utilizzata per definire i tipi di comandi processati dai server di ConText. Esistono più categorie di server di ConText. L'insieme di categorie gestite da un server di ConText definisce la tipologia del server. Le categorie disponibili sono: Q (Query), D (DDL), M (DML) e L (Linguistica). Vedere il Capitolo 30.

CATENA, BLOCCHI IN

Un blocco in catena è un secondo o successivo blocco di ORACLE destinato a memorizzare dati di tabelle, quando il blocco allocato in partenza ha esaurito lo spazio e le righe in quel blocco si ampliano a causa di operazioni di update. Viene più spesso utilizzato per dati di tabelle, ma anche i dati degli indici possono essere memorizzati a catena. I blocchi in catena hanno conseguenze sulle prestazioni, perciò quando compaiono, il parametro che definisce lo spazio PCTFREE può risultare troppo basso.

CATENA, RIGA IN

Una riga in catena è una riga che viene memorizzata in più di un blocco di database e che quindi è composta da più parti. Righe lunghe (o dati di tipo LONG), la cui lunghezza è maggiore della dimensione di un blocco possiedono sempre vari pezzi di riga. In ORACLE, il comando ANALYZE può identificare le righe in catena e fornire informazioni statistiche sulla loro entità. Vedere ANALYZE.

CEIL

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE FLOOR; NUMERI, FUNZIONI

SINTASSI CEIL(*valore*)

DESCRIZIONE CEIL è il più piccolo intero maggiore o uguale a *valore*.

ESEMPIO CEIL(2) = 2
 CEIL(1.3) = 2
 CEIL(-2) = -2
 CEIL(-2.3) = -2

CHANGE

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE APPEND, DEL, EDIT, LIST, Capitolo 5

SINTASSI C[HANGE] /*vecchio testo/nuovo testo/*

DESCRIZIONE CHANGE è una funzionalità dell'editor a riga di comando di SQL*PLUS.

Consente di modificare un *vecchio testo* in un *nuovo testo* all'interno della riga corrente del buffer in uso (la riga 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 ricorrenza, 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 / 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 qualsiasi carattere successivo alla parola CHANGE (diverso dallo spazio) sia il delimitatore.

ESEMPI Se questa è la riga corrente del buffer in uso:

where Mansione in ('Fabbro', 'Scavatore fosse', 'Buon trebbiatore')

allora:

C /Fabbro/Aratore

esegue la seguente modifica:

where Mansione in ('Aratore', 'Scavatore fosse', 'Buon trebbiatore')

Questo comando:

C ?Fabbro',...?Aratore')

(si noti il ? utilizzato come delimitatore) produce la seguente modifica:

where Mansione in ('Aratore')

Quest'altro:

C /Scavatore...Buon, '/Primo

produce l'output seguente:

where Mansione in ('Fabbro', 'Primo trebbiatore')

CHAR, TIPO DI DATI

Vedere DATI, TIPI

CHARTOROWID

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CONVERSIONE, FUNZIONI; ROWIDTOCHAR

SINTASSI CHARTOROWID(*stringa*)

DESCRIZIONE Il nome di questa funzione 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.

NOTA ORACLE esegue questo tipo di conversione automaticamente, perciò questa funzione non è realmente necessaria. Si tratta di uno strumento di debugging che ha trovato un proprio modo di disponibilità generale.

CHECKPOINT

Un checkpoint è un punto in cui, su base di sessione, i blocchi modificati vengono scritti nel database. Un checkpoint si verifica quando il numero di file redo log scritti egualia LOG_CHECKPOINT_INTERVAL e, inoltre, quando viene riempito un file redo log in linea.

CHIAMATE RICORSIVE

Una chiamata ricorsiva è una chiamata annidata dell'RDBMS; Ad esempio, le informazioni di auditing vengono registrate utilizzando chiamate ricorsive. Durante il normale funzionamento del database, magari durante un aggiornamento, viene ese-

guita un'altra operazione che scrive il report, gestisce l'auditing e le altre informazioni fondamentali.

CHIAVE

Una chiave è una colonna utilizzata per identificare delle righe; non è equivalente a 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 sulle 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 univocamente ciascuna riga di una tabella.

CHIAVE UNICA

Una chiave unica è una colonna che deve contenere valori unici per ciascuna riga della tabella. Vedere CHIAVE, CHIAVE PRIMARIA e VINCOLI DI INTEGRITÀ.

CHR

TIPO Funzione SQL

PRODOTTI Tutti

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 254, poiché il valore ASCII di un carattere è un intero tra 0 e 254). I valori tra lo 0 e il 127 possiedono standard ben definiti. I valori superiori al 127 (chiamati insieme esteso di caratteri ASCII) tendono a differire da nazione a nazione, a seconda dell'applicazione e del costruttore di computer. La lettera A, ad 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.

ESEMPI select CHR(77), CHR(46), CHR(56) from DUAL;

C C C

- - -

M . 8

CICLI

È possibile utilizzare dei cicli per gestire diversi 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 gestire record multipli a partire da un cursore. Il ciclo a cursore in assoluto più comune è il ciclo FOR. Vedere CICLI A CURSORE e il Capitolo 22 per dettagli riguardanti l'uso di logiche di esecuzione semplici e dinamiche.

CICLI A CURSORE

Nei cicli FOR, le iterazioni vengono eseguite per un numero specifico di volte. In un ciclo FOR associato a un cursore, vengono utilizzati i risultati di una query per determinare in modo dinamico quante volte il ciclo debba essere eseguito. In un ciclo FOR associato a un cursore, l'apertura, l'operazione di fetch e la chiusura dei cursori viene effettuata in modo implicito, non è necessario dichiarare in modo esplicito queste azioni.

Nel listato seguente, viene mostrato un ciclo FOR di questo tipo che interroga la tabella RAGGI_VAL e inserisce dei record nella tabella AREE.

```
declare
    pi      constant NUMBER(9,7) := 3.1415926;
    area   NUMBER(14,2);
    cursor rag_cursor is
        select * from RAGGI_VAL;
    rag_val rag_cursor%ROWTYPE;
begin
    for rag_val in rag_cursor
    loop
        area := pi*power(rag_val.raggio,2);
        insert into AREE values (rag_val.raggio, area);
    end loop;
end;
/

```

In un ciclo FOR associato a un cursore, non vi è alcun comando open o fetch. Il comando:

```
for rag_val in rag_cursor
```

apre in modo implicito il cursore “rag_cursor” e assegna un valore alla variabile “rag_val”. Quando non vi sono più record nel cursore, il ciclo viene terminato e il cursore viene chiuso. In un ciclo FOR associato a cursore, non vi è necessità di un comando close. Vedere il Capitolo 22 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

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE BREAK, COLUMN, COMPUTE

SINTASSI CL[EAR] *opzione*

DESCRIZIONE CLEAR azzerà tutte le opzioni.

BRE[AKS] elimina i break impostati tramite il comando BREAK.

BUFF[ER] azzerà il buffer corrente.

COL[UMNS] azzerà 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 di SQL.

TIMI[NG] elimina tutte le aree di timing create dal comando TIMING.

ESEMPI Per azzerare i calcoli, si esegua 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

Vedere DATI, TIPI e Capitolo 27.

CLOSE

TIPO Istruzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE DECLARE, FETCH, FOR, OPEN, Capitolo 22

SINTASSI CLOSE *cursor;*

DESCRIZIONE CLOSE chiude il cursore specificato e ne rilascia le risorse in modo che ORACLE le possa riutilizzare altrove. *cursor* deve essere il nome di un cursore attualmente aperto.

Anche se un cursore è stato chiuso, la sua definizione non risulta persa ed è possibile eseguire nuovamente OPEN *cursor* in quanto precedentemente è stato dichiarato in modo esplicito. Un ciclo FOR apre in modo implicito un cursore dichiarato. Vede CICLI A CURSORI.

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. Vede CREATE CLUSTER e il Capitolo 19.

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. Ad esempio, le due tabelle LAVORATORE e COMPITOLAVORATORE possono essere asso-

ciate a cluster sulla colonna Nome. La chiave di un cluster è la stessa cosa della colonna di un cluster.

CLUSTER, COLONNE

Vedere CLUSTER, CHIAVE.

CLUSTER, INDICE

L'indice di un cluster viene creato manualmente dopo la creazione del cluster e prima che qualsiasi istruzione DML (cioè 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 di hash per indicizzare la chiave primaria. *Vedere CLUSTER DI HASH.*

CLUSTER DI HASH

Un cluster di hash è un cluster memorizzato tramite una chiave di hash invece che attraverso una chiave numerica. Una chiave di hash è un valore calcolato a partire dai valori delle chiavi che rappresentano la locazione su disco. Una chiave indice necessita dell'intervento di ORACLE per garantire l'archiviazione del cluster, una chiave di hash calcola automaticamente la locazione.

CMDSEP

Vedere SET.

COALESCE

Questa opzione consente di unire estensioni libere adiacenti in un'unica estensione. Ad esempio, se due estensioni di 100 blocchi sono vicine l'una all'altra all'interno di un tablespace, possono essere riunite in una singola estensione di 200 blocchi. Il processo di background SMON consente di unire spazi liberi all'interno di tablespace il cui valore pctincrease di default è diverso da zero. È possibile riunire manualmente dello spazio libero all'interno di un tablespace tramite l'opzione coalesce del comando alter tablespace. *Vedere ALTER TABLESPACE.*

CODA DI AUDITING

Una coda di auditing è una tabella del database che viene scritta quando è abilitato il comando di auditing (tramite il parametro di init.ora) e sono state selezionate dagli utenti o dal DBA le relative opzioni. Una tabella di base di proprietà di SYS contiene tutte le righe della coda di auditing.

COERENZA

Consistenza è un termine generale e un concetto riferito ai database che sta a indicare che tutti i dati tra loro correlati devono essere aggiornati assieme nell'ordine corretto e che, se vi sono dati ridondanti, devono essere tutti tra di loro congruenti.

COERENZA DI LETTURA

La coerenza di lettura è uno stato che garantisce che tutti i dati di un'istruzione o di una transazione siano compatibili con 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. Ad esempio, in una tabella contenente i dati dei dipendenti di una ditta, tutte le età dei dipendenti costituiscono una colonna. *Vedere RIGA.*

COLONNA, VINCOLO

Un vincolo su una colonna è un vincolo integrità assegnato a una specifica colonna di una tabella. *Vedere VINCOLO DI INTEGRITÀ.*

COLONNA DI UNIONE

Una colonna di unione è una colonna utilizzata nell'unione di una tabella con un'altra. Queste colonne possono essere identificate utilizzando la clausola where e specificando la relazione che le lega alle tabelle.

COLUMN

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE ALIAS, Capitoli 5 e 13

SINTASSI COL[UMN] {colonna | espressione}
 [ALI[AS] alias]
 [CLE[AR] | DEF[AULT]]
 [FOR[MAT] formato]
 [HEA[DING] testo
 [JUS[TIFY] {L[EFT] | CENTER} | C[ENTRE] | RIGHT]]]
 [LIKE {espressione | etichetta}]
 [NEWL[INE]]
 [NEW V[ALUE] variabile]
 [NUL[L] testo]
 [NOPRI[NT] | PRI[NT]]
 [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, che 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 attivo l'utilizzo più 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 a un'espressione utilizzata nell'istruzione select. Se viene utilizzata un'espressione, quest'ultima deve 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 funziona. 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 se-
quenziali), un comando COLUMN per quel nome viene applicato a ciascuna colonna
con quel nome. Per evitare ciò è sufficiente assegnare alle colonne, all'interno dell'
istruzione select, 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.

FORMAT specifica il formato di visualizzazione della colonna. Il formato deve
essere un valore letterale quale A25 o 990.99.

Senza un formato specifico, l'ampiezza della colonna coincide con la lunghezza
come definito nella tabella.

L'ampiezza di una colonna di tipo LONG ha un valore di default pari a SET
LONG. È possibile impostare l'ampiezza sia dei campi CHAR che LONG normali tra-
mite un formato del tipo FORMAT An, dove *n* rappresenta un intero con la nuova di-
mensione della colonna.

L'ampiezza di una colonna numerica possiede come default il valore di SET NU-
MWIDTH, 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 co-
lumn format:

| FORMATO | RISULTATO |
|--------------|--|
| 9999990 | Il conteggio dei nove o zero determina le cifre massime che possono essere visualizzate. |
| 9,999,999.99 | Le virgole e i decimali vengono specificati nel modello mostrato. Se il valore è zero può non venire visualizzato nulla. |
| 999990 | Visualizza uno zero se il valore è zero. |
| 099999 | Visualizza numeri con zeri iniziali. |
| \$99999 | Segno di dollaro inserito all'inizio di ciascun numero. |
| B99999 | Se il valore è zero, non viene visualizzato nulla. Questo è il default. |
| 99999MI | Se il numero è negativo, il segno meno segue il numero. Il default è il segno negativo a sinistra. |
| 99999PR | Numeri negativi racchiusi tra < e >. |
| 99999EEEE | La visualizzazione risulta con notazione scientifica. Devono essere esattamente quattro E. |
| 999V99 | Moltiplica il numero per 10 ⁿ dove <i>n</i> rappresenta il numero di cifre a destra di V. 999V99 traduce 1234 in 123400. |
| DATE | Formatta una colonna numerica nel formato data del calendario Giuliano, quale MM/DD/YY |

HEADING riassegna un'etichetta all'intestazione di una colonna. Il default è rappresentato dal nome della colonna o dall'espressione. Se il testo possiede spazi o caratteri di punteggiatura, deve essere racchiuso tra apici singoli. Il carattere **HEADSEP** (solitamente ‘|’) all'interno di un testo fa in modo che SQL*PLUS inizi una nuova riga. 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 qualsiasi 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 13 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**. Il default 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 specifica 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 **TRUNC** controlla 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. **TRUNC** tronca il valore all'ampiezza della definizione della colonna.

Vedere i Capitoli 5 e 13 per esempi al riguardo e **SET RECSEP** per esempi sugli effetti del carattere separatore di record.

COMANDI DI SISTEMA PRIVILEGIATI

I comandi di sistema privilegiati sono un sottoinsieme di comandi SQL che richiede non solo l'accesso alle utilità DBA ma anche un particolare account nel sistema operativo. Questi comandi richiedono il più elevato livello di sicurezza possibile.

COMANDO

Vedere **ISTRUZIONE**.

COMANDO, RIGA DI

La riga di comando è la riga sullo schermo del computer dove vengono inseriti i comandi.

COMMENT

PRODOTTO SQL*PLUS

VEDERE ANCHE VISTE DEL DIZIONARIO DI DATI, Capitolo 32

SINTASSI COMMENT ON {TABLE *tabella* | COLUMN *tabella.colonna*} IS *testo*

DESCRIZIONE COMMENT inserisce nel dizionario di dati il testo di commento relativo a una tabella o a una colonna. Per informazioni su come vedere e impostare i commenti, vedere il Capitolo 32.

Per eliminare un commento dal database è sufficiente impostarlo al valore NULL (set *testo* to "").

COMMIT

Questa espressione significa apportare modifiche ai dati (tramite insert, update e delete) in modo permanente. Prima che le modifiche vengano registrate, 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)

TIPO Comando SQL

PRODOTTO Precompilatori

VEDERE ANCHE ROLLBACK, SAVEPOINT, SET TRANSACTION

SINTASSI EXEC SQL [AT *database*] COMMIT [WORK] [RELEASE]

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 blocco 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 opzionale 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 che locale.

COMMIT (forma 2, istruzione PL/SQL)

TIPO Istruzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE ROLLBACK, SAVEPOINT

SINTASSI COMMIT [WORK];

DESCRIZIONE COMMIT rende effettive le modifiche apportate al database dal momento in cui è stato eseguito l'ultimo COMMIT, sia隐式的还是显式的. WORK è opzionale e non ha effetti sull'utilizzo.

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.

CONFRONTO

TIPO Definizione

PRODOTTO SQL

VEDERE ANCHE GROUP BY, INDEX, ORDER BY, Capitolo 8

DESCRIZIONE La sequenza di confronto è l'ordine in cui caratteri, numeri e simboli vengono ordinati a causa di una clausola order by o group by. Queste sequenze differiscono a seconda della sequenza di confronto del sistema operativo del computer o del linguaggio 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. Lo spagnolo "ll" si trova in una certa posizione della sequenza di caratteri. Nonostante queste differenze, è possibile applicare sempre le seguenti regole:

- un numero con un valore più grande viene considerato "maggiori" di quello con il valore più piccolo; tutti i numeri negativi sono minori di quelli positivi; quindi, -10 è minore di 10; -100 è minore di -10;
- una data più recente è considerata maggiore di una più indietro nel tempo.

Le stringhe di caratteri vengono confrontate posizione per posizione, a partire dall'estremità più a sinistra della stringa, fino al primo carattere differente. La stringa che possiede il carattere "maggiori" in quella posizione viene considerata la più grande. Un carattere risulta maggiore di un altro se, nella sequenza di confronto del computer, compare dopo. Solitamente, ciò significa che B è maggiore di A, anche se il confronto tra il valore di A con quello di a o con il numero 1, fornisce risultati diversi da computer a computer.

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 della stessa lunghezza, vengono considerate uguali.

Con stringhe di tipo CHAR, la stringa più breve viene riempita con spazi fino alla lunghezza della stringa più lunga. Se, dopo quest'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 VARCHAR.

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'.

COMPUTE

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE BREAK, FUNZIONI DI GRUPPO

SINTASSI COMP[UTE] [AVG|COU[N]T|MAX|IMUM|MIN|NUM|BER|STD|SUM|VAR[IANCE]]...
 LABEL *nome etichetta*
 OF {espressione} [, espressione]...
 ON [espressione | PAGE | 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 di default.

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, pagina o report. Vedere BREAK per i dettagli riguardo 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 virgole, 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 Elemento skip 2 on report skip 1
compute sum avg max of Importo Aliquota on Elemento report
```

```
select Elemento, Aliquota, Importo
```

```
from REGISTRO
order by Elemento;
```

Si noti l'importanza di order by Elemento per il corretto funzionamento.

COMPUTER REMOTO

Un computer remoto è un calcolatore di rete diverso da quello da cui si sta effettuando l'interrogazione.

CONCAT

Vedere SET, ||.

CONCATENAZIONE

La concatenazione è l'unione di stringhe, rappresentata dall'operatore “||” (da non confondere con le unioni di tabelle). Ad esempio, la concatenazione delle stringhe ‘ABC’ e ‘XYZ’ viene indicata con ‘ABC’||‘XYZ’ e dà come risultato ‘ABCXYZ’.

CONDIZIONE

Una condizione è un'espressione il cui valore viene valutato TRUE o FALSE, come in Eta > 65.

CONNECT (Forma 1)

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE COMMIT, DISCONNECT, Capitolo 21

SINTASSI CONNECT [utente[/password] [@database];

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 sconnessione 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* esegue la connessione al database specificato, che può risiedere sull'host dell'utente o su un altro computer collegato tramite SQL*NET.

CONNECT (forma 2, interno SQL)

TIPO Comando SQL

PRODOTTO Precompilatori

VEDERE ANCHE COMMIT, DECLARE DATABASE, Capitolo 21

SINTASSI EXEC SQL CONNECT :password_utente

[AT *database*]

[USING :stringa_connessione]

EXEC SQL CONNECT :utente IDENTIFIED BY :password

[AT *database*]

[USING :stringa_connessione]

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 del-

l'utente di ORACLE e la password separati da una barra (/). In alternativa, :utente e :password possono essere separati utilizzando il secondo formato.

AT viene utilizzato per specificare un database diverso da quello di default attivo per l'utente in questione. Si tratta di una clausola richiesta per raggiungere qualsiasi database diverso da quello di default per l'utente. Questo nome può essere utilizzato successivamente in altre istruzioni SQL con AT. Questo database deve essere prima identificato con DECLARE DATABASE. USING specifica una stringa di SQL*NET opzionale (quale il nome di un nodo) utilizzata durante la connessione. Senza la stringa USING, la connessione avviene con il database di default dell'utente, indipendentemente dal database specificato alla riga AT.

CONNECT BY

TIPO Operatore di comando SQL

PRODOTTI Tutti

VEDERE ANCHE Capitolo 12

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 resoconti sulle gerarchie di eredità in dati strutturati ad albero, quali 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, a seconda della localizzazione di PRIOR);
- una specifica nella clausola CONNECT BY (in particolare un segno di diverso anziché di uguale) elimina sia un elemento singolo che tutti i suoi discendenti;
- CONNECT BY non può essere utilizzato con un'unione di tabella nella clausola where.

ESEMPIO select Mucca, Toro, LPAD(' ',6*(Level-1))||Figlio Figlio,
Sex, Datanascita
from ALLEVAMENTO
connect by Figlio = PRIOR Vacca
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. Vedere il Capitolo 12 per una trattazione più estesa di CONNECT BY.

CONNETTERSI

Connettersi significa identificarsi a ORACLE tramite il proprio nome utente e la propria password, allo scopo di accedere al database.

CONTAINS

CONTAINS viene utilizzato per valutare ricerche di testi che utilizzano ConText di ORACLE. Vedere OPERATORI DI RICERCA TESTUALE e il Capitolo 29.

CONTEMPORANEITÀ

Contemporaneità è un termine generale che indica l'accesso allo stesso dato da parte di più utenti. Nel software del database, la contemporaneità 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.

CONTEXT

ConText (anche indicato col nome di ORACLE ConText Option) è un motore di ricerca dei testi disponibile all'interno del server di ORACLE. ConText consente di eseguire ricerche di testi, incluse le corrispondenze fuzzy, le espansioni di radice e le ricerche di frasi. Vedere i Capitoli 29 e 30.

CONTEXT, SERVER

Un server di ConText è un server in background che partecipa alla risoluzione delle query coinvolte nelle ricerche di testi. Per amministrare e utilizzare ConText, è necessario che siano abilitati i server di ConText. Vedere il Capitolo 30.

CONVERSIONE, FUNZIONI

TIPO Funzioni SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; NUMERI, FUNZIONI

DESCRIZIONE Di seguito è riportato un elenco in ordine alfabetico di tutte le funzioni di conversione e trasformazione attualmente disponibili in SQL di ORACLE. Ciascuna di esse è presente sotto il proprio nome all'interno di questa guida, con informazioni riguardanti il formato appropriato e l'utilizzo.

Nomi di funzioni e loro utilizzo

`CHARTOROWID(stringa)`

CHARTOROWID sta per CHARacter TO ROW IDentifier (da carattere a identificatore di riga). Consente di modificare una stringa di caratteri in modo che funzioni da identificatore di riga interno di ORACLE o ROWID.

`CONVERT(stringa,[ins_destinazione,[ins_sorgente]])`

CONVERT converte i caratteri contenuti in stringa da uno standard di rappresentazione a bit a un'altro, quale, ad esempio, dall'US7ASCII al WE8DEC.

`DECODE(valore,if1,then1,if2,then2,if3,then3,...,else)`

DECODE decodifica una stringa di caratteri, una data (DATE) o un numero (NUMBER) in una serie di varie stringhe differenti, date o numeri, in base a un

valore. Si tratta di una funzione di tipo IF, THEN, ELSE molto potente. Ad essa è dedicato il Capitolo 16.

`HEXTORAW(stringa_esadecimale)`

`HEXTORAW` sta per HEXadecimAl TO RAW. Modifica una stringa di caratteri di numeri esadecimali in binario.

`RAWTOHEX(stringa_binaria)`

`RAWTOHEX` sta per RAW TO HEXadecimAl. Modifica una stringa di numeri binari in una stringa di caratteri di numeri esadecimali.

`REPLACE(stringa, if [,then])`

`REPLACE` restituisce *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 ricorrenze di *if*. Vedere `TRANSLATE`.

`ROWIDTOCHAR(RowId)`

`ROWIDTOCHAR` sta per ROW IDentifier TO CHARacter (da identificatore di riga a carattere). Modifica un identificatore di riga interno di ORACLE o RowId, in modo che agisca come una stringa di caratteri.

`TO_CHAR(valore)`

`TO_CHAR` sta per TO CHARacter (in carattere). Converte un numero in modo che possa essere utilizzato come stringa di caratteri.

`TO_DATE(stringa, ['formato'])`

`TO_DATE` converte un elemento di tipo NUMBER, CHAR o VARCHAR2 in un elemento che funziona come un tipo DATE (un tipo di dato speciale di ORACLE).

`TO_LABEL(stringa[, 'formato'])`

`TO_LABEL` converte una stringa di caratteri in un valore corrispondente al tipo di dato RAW MLSLABEL. Vedere *Trusted ORACLE Administrator's Guide* per ulteriori informazioni.

`TO_MULTI_BYTE(stringa)`

`TO_MULTI_BYTE` converte una stringa di caratteri contenente caratteri a singoli byte nei corrispondenti caratteri a più byte. Se un certo carattere non ha un equivalente a più byte, compare come carattere a un unico byte. Questa funzione consente di riunire, in una stringa, caratteri a un singolo byte o a più byte.

`TO_NUMBER(stringa)`

`TO_NUMBER` converte una stringa in un elemento che funziona come un numero.

`TO_SINGLE_BYTE(stringa)`

`TO_SINGLE_BYTE` converte una stringa di caratteri con caratteri a più byte nei corrispondenti caratteri a un singolo byte. Se un certo carattere non possiede l'equivalente a un unico byte, compare come carattere a più byte. Questa funzione consente di riunire in una certa stringa caratteri a un unico byte con caratteri a più byte.

TRANSLATE(*stringa,if,then*)

TRANSLATE traduce i caratteri contenuti in una stringa in caratteri differenti.

CONVERT

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CONVERSIONE, FUNZIONI; *ORACLE Server Administrator's Guide*

SINTASSI CONVERT(*stringa,[ins_destinazione,[ins_sorgente]]*)

DESCRIZIONE CONVERT converte i caratteri di una *stringa* da uno standard di rappresentazione a bit in un altro, come, ad esempio, dal WE8DEC. Quest'operazione viene normalmente effettuata 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.

| | |
|--------------|--|
| F7DEX | Set a 7-bit ASCII di DEC per la Francia |
| US7ASCII | Set US 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 |

COPY

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE CREATE DATABASE LINK, Capitolo 21

SINTASSI COPY [FROM *utente/password@database*]
 [TO *utente/password@database*]
 {APPEND | CREATE | INSERT | REPLACE}
 tabella[(colonna [,colonna]...)]
 USING *query*;

DESCRIZIONE COPY copia una tabella in un'altra tabella che risiede su un computer differente, attraverso SQL*NET. FROM specifica il nome dell'utente, la password e un database della tabella sorgente, mentre TO rappresenta la tabella di destinazione. Sia FROM che TO possono essere omessi, nel qual caso viene utilizzato, al posto della clausola mancante, il database di default dell'utente. Il database sorgente 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, la crea. CREATE chiede di creare la tabella di destinazione; se esiste già viene visualizzato 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 viene visualizzato 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 sorgente; 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 specificate, il loro numero deve essere lo stesso di quello delle colonne della query. Se non viene specificata alcuna colonna, alle colonne copiate nella tabella di destinazione, vengono assegnati gli stessi nomi delle colonne contenute nella tabella sorgente. *query* identifica la tabella sorgente 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 In questo esempio vengono copiati i record dalla tabella REGISTRO a EDMESTON nel database con cui è collegato l'utente di SQL*PLUS locale. La tabella TOTALE_REGISTRO viene creata dalla copia. Solo due colonne vengono copiate e una di esse è una colonna calcolata, una somma. Queste colonne, una volta giunte a destinazione, vengono rinominate Oggetto e Somma. Si noti l'utilizzo, richiesto, del trattino (-) alla fine di ciascuna riga.

```
copy from GIORGIO/TAL@EDMESTON -
create TOTALE_REGISTRO (Oggetto, Somma) -
using select Oggetto, SUM(Somma) -
      from REGISTRO -
group by Oggetto;
```

COPYCOMMIT

Vedere SET.

CORRENTE, BUFFER

Il buffer corrente è il buffer su cui possono agire i comandi di modifica di SQL*PLUS e che può essere salvato in un file tramite il comando SAVE. *Vedere CHANGE.*

CORRENTE, RIGA

La riga corrente è la riga all'interno del buffer corrente su cui possono agire i comandi dell'editor della riga di comando di SQL*PLUS. *Vedere CORRENTE, BUFFER e CHANGE.*

COS

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ACOS, ASIN, ATAN, ATAN2, COSH, EXP, LN, LOG, SIN, SINH, TAN, TANH

SINTASSI COS(*valore*)

DESCRIZIONE COS restituisce il coseno di *valore*, un angolo espresso in radianti. Per convertire in radianti la misura di un angolo espressa in gradi occorre moltiplicarlo per pi/10.

ESEMPIO COL COSENO_180 HEADING "Coseno di 180 gradi"
SELECT COS(180*3.14159/80) COSENO_180 FROM DUAL;
Coseno di 180 gradi

-1

COSH

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ACOS, ASIN, ATAN, ATAN2, COS, EXP, LN, LOG, SIN, SINH, TAN, TANH

SINTASSI COSH(*valore*)

DESCRIZIONE COSH restituisce il coseno iperbolico di *valore*.

ESEMPIO COL ICOSENO HEADING "Coseno iperbolico di 0"
SELECT COSH(0) ICOSENO FROM DUAL;
Coseno iperbolico di 0

1

COUNT

TIPO Funzione SQL di gruppo

PRODOTTI Tutti

VEDERE ANCHE FUNZIONI DI GRUPPO, Capitolo 7

SINTASSI COUNT([DISTINCT] *espressione* | *)

DESCRIZIONE COUNT conta il numero di righe, restituite da una query, dove *espressione* non è NULL. L'opzione DISTINCT fa in modo che vengano contate solo le righe distinte non NULL. Con il simbolo *, vengono contate tutte le righe, sia NULL che non NULL.

CREATE CLUSTER

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE INDEX, CREATE TABLE, Capitolo 19

SINTASSI CREATE CLUSTER [*utente*.]*cluster*
(*colonna tipodato*
[,*colonna tipodato*]...)
[INITTRANS *intero*]
[MAXTRANS *intero*]
[PCTFREE *intero*]
[PCTUSED *intero*]
[SIZE *intero*]
[STORAGE *registra*]
[TABLESPACE *tablespace*]
[INDEX | [HASH IS *colonna*] HASHKEYS *intero*]
[NOCACHE|CACHE]
[NOPARALLEL|PARALLEL [DEGREE *n*] [INSTANCES *n*]]

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 rende effettive le modifiche pendenti nei database. Richiede almeno una colonna cluster da ciascuna tabella. Devono avere la stessa dimensione ed essere dello stesso tipo, ma non è richiesto che siano omonime. Per le tabelle di un cluster, le righe con gli stessi valori della colonna cluster vengono riunite assieme nella stessa area del disco, nello stesso blocco o negli stessi blocchi logici. In questo modo, le prestazioni possono migliorare nel caso in cui le colonne cluster siano colonne attraverso cui le tabelle vengono solitamente unite. Vedere il Capitolo 19.

Ogni valore distinto nella colonna cluster viene registrato solo una volta, indipendentemente dal fatto che ricorra più volte nelle tabelle e nelle righe. Normalmente, 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. La lunghezza massima di tutte le colonne cluster combinate (per un comando CLUSTER) è di 239 caratteri. Le tabelle con colonne di tipo LONG non possono essere associate a cluster. *cluster* è il nome assegnato al cluster. *colonna* e *tipodato* seguono il metodo di CREATE TABLE, fatta eccezione per il fatto che NULL e NOT NULL non possono essere specificati. Tuttavia, nell'istruzione CREATE TABLE vera e propria, almeno una colonna di un cluster deve essere NOT NULL. SIZE imposta la dimensione in byte di un blocco logico (diversa dal blocco fisico). SPACE rappresenta la collocazione iniziale su disco del cluster, come viene utilizzata in CREATE TABLE.

SIZE dovrebbe essere la quantità media di spazio necessaria per registrare tutte le righe da tutte le tabelle contenute in cluster che risultano associate a un'unica chiave del cluster. Un valore di SIZE piccolo può far lievitare il tempo necessario per accedere alle tabelle nel cluster, ma può ridurre l'utilizzo di spazio su disco. SIZE dovrebbe essere un conveniente divisore 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 del cluster prima di inserire i dati. Se si specifica la forma di cluster di hash, tuttavia, non è necessario (e non è nemmeno possibile) creare un indice sulla chiave del cluster. Al contrario, ORACLE utilizza una funzione di hash per registrare 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 associate alla chiave del cluster. La clausola HASHKEYS in realtà crea il cluster di hash e specifica il numero di valori hash, arrotondati al numero primo più vicino. Il valore minimo è 2.

Quando vengono letti i blocchi dal database, ORACLE li memorizza nella SGA. ORACLE gestisce un elenco (LRU) dei blocchi utilizzati meno recentemente; quando nella SGA viene richiesto uno spazio aggiuntivo, questi blocchi di dati vengono rimossi dalla SGA.

CACHE modifica questo funzionamento specificando che i blocchi per il cluster in questione devono essere collocati all'estremità dei blocchi utilizzati più di recente

nell'elenco LRU. Quindi, i blocchi restano nella SGA più a lungo. Questa opzione risulta utile se il cluster contiene piccole tabelle di ricerca. NOCACHE (l'opzione di default) riporta il cluster al normale funzionamento dell'elenco LRU.

PARALLEL, assieme a DEGREE e INSTANCES, specifica le caratteristiche parallele del cluster (per database che utilizzano l'opzione Parallel Query). DEGREE specifica il numero di server di query da utilizzare; INSTANCES specifica il modo in cui il cluster deve essere suddiviso tra le istanze di un server parallelo per l'elaborazione di query in parallelo. Un intero *n* indica che il cluster deve essere suddiviso tra il numero specificato di istanze disponibili.

CREATE CONTROLFILE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER DATABASE, CREATE DATABASE

| | |
|-----------------|---|
| SINTASSI | CREATE CONTROLFILE [REUSE] [SET] DATABASE <i>database</i> |
| | LOGFILE [GROUP <i>intero</i>] <i>file</i> |
| | [, [GROUP <i>intero</i>] <i>file</i>] ... |
| | {RESETLOGS NORESETLOGS} |
| | DATAFILE <i>file</i> [, <i>file</i>] |
| | [MAXLOGFILES <i>intero</i>] |
| | [MAXLOGMEMBERS <i>intero</i>] |
| | [MAXLOGHISTORY <i>intero</i>] |
| | [MAXDATAFILES <i>intero</i>] |
| | [MAXINSTANCES <i>intero</i>] |
| | [ARCHIVELOG NOARCHIVELOG] |

DESCRIZIONE Il comando CREATE CONTROLFILE ricrea un file di controllo nel caso in cui l'utente abbia perso il proprio file di controllo corrente a causa di un inconveniente tecnico, oppure nel caso si voglia modificare il nome del proprio database o una delle opzioni per il file di log di un file di dati. Prima di eseguire questa operazione, è opportuno effettuare un salvataggio completo dei file del database.

L'opzione REUSE consente di riutilizzare i file di controllo esistenti anziché visualizzare un errore. L'opzione SET modifica il nome del database, specificato dalla clausola del database. La linea LOGFILE specifica i gruppi di file redo log, ognuno dei quali deve esistere. L'opzione RESETLOGS e la sua controparte NORESETLOGS comunicano a ORACLE, rispettivamente, 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 file redo log redo che possono essere creati. L'opzione MAXLOGMEMBERS specifica il numero di copie per un gruppo di redo log. MAXLOGHISTORY specifica invece il numero di gruppi di file redo log archiviati per il server parallelo. 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 montare e aprire il database. Le opzioni ARCHIVELOG e NOARCHIVELOG, rispettivamente, abilitano e disabilitano l'archiviazione dei file redo log. Il comando CREATE CONTROLFILE necessario per un database esistente, può essere generato attraverso il comando ALTER DATABASE BACKUP CONTROLFILE TO TRACE.

CREATE DATABASE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER DATABASE, AVVIO, CHIUSURA, CREATE CONTROLFILE, CREATE ROLLBACK SEGMENT, CREATE TABLESPACE

SINTASSI

```

CREATE DATABASE [database]
  [CONTROLFILE REUSE]
    [LOGFILE [GROUP intero] definizione_file
      [, [GROUP intero] definizione_file]... ]
    [MAXLOGFILES intero]
    [MAXLOGMEMBERS intero]
    [MAXLOGHISTORY] intero
      [DATAFILE definizione_file [,definizione_file]... ]
      [AUTOEXTEND definizione_file [,definizione_file] [ON | OFF]
        [NEXT intero [K | M]]
        [MAXSIZ
          [UNLIMITED | intero [K | M]]]]
    [MAXDATAFILES intero]
    [MAXINSTANCES intero]
    [ARCHIVELOG | NOARCHIVELOG]
    [EXCLUSIVE]
  {CHARACTER SET insiemecar}

```

DESCRIZIONE *database* è il nome del database, di lunghezza pari o inferiore a otto caratteri. Un nome di default per il database viene specificato dal parametro DB_NAME di init.ora. *definizione_file* specifica i nomi e le dimensioni per LOGFILE e DATAFILE:

```
'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 riscrive i file di controllo esistenti definiti dal parametro CONTROL_FILES di init.ora.

LOGFILE definisce il nome dei file da utilizzare come file redo log. Se questo parametro non viene utilizzato, ORACLE ne crea due per default. MAXLOGFILES riscrive il parametro LOG_FILES di init.ora e definisce il numero massimo di file redo log che possono essere creati per il database. Questo numero non può essere successivamente incrementato. Il valore minimo è 1. Il massimo è 256. Un numero alto crea solo un file di controllo più grande.

DATAFILE specifica i nomi dei file che il database stesso utilizzerà. Questi file vengono automaticamente collocati nel tablespace SYSTEM. Se si omette questa clausola, ORACLE crea per default un unico file. Il nome e la dimensione di questo file differiscono a seconda del sistema operativo. MAXDATAFILES imposta il valore limite superiore assoluto per i file che possono essere creati per il database in questione e riassegna il parametro DB_FILES di init.ora. Il valore massimo è 255. Il valore di default è 32. Un numero alto crea solo un file di controllo più grande. Quando viene abilitata (ON) l'opzione AUTOEXTEND per un file di dati, il file di dati

viene dinamicamente esteso della quantità necessaria con incrementi di dimensione pari a NEXT, fino a un valore massimo pari a MAXSIZE (o UNLIMITED, illimitato).

MAXINSTANCES riscrive il parametro INSTANCES di init.ora e imposta il numero massimo di istanze che possono montare e aprire il database. Il numero massimo è pari a 255.

ARCHIVELOG e NOARCHIVELOG definisce il modo in cui i file redo log vengono utilizzati quando il database viene creato per la prima volta. NOARCHIVELOG è il default e significa che i file redo log vengono riutilizzati senza salvare il loro contenuto altrove. In questo modo, viene garantito il ripristino dell'istanza, ma non il ripristino dei dati a fronte di un malfunzionamento del sistema, quale un crash del disco. ARCHIVELOG forza 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. Gestisce anche il ripristino dell'istanza. Questo parametro può essere azzerato tramite il comando ALTER DATABASE.

L'opzione MAXLOGMEMBERS specifica il numero massimo di copie di un gruppo di file redo log. L'opzione MAXLOGHISTORY specifica il numero massimo di file redo log archiviati, utile solo per il server parallelo quando si stanno archiviando i file di log. L'opzione CHARACTER SET specifica l'insieme di caratteri utilizzato per registrare i dati, che dipende dal sistema operativo.

EXCLUSIVE è interamente opzionale e viene utilizzato qui come promemoria del fatto che tutti i database sono stati creati in modo da consentire a una sola istanza di accedere al database. Per consentire a più istanze di accedere al database, è necessario utilizzare i comandi ALTER DATABASE DISMOUNT e ALTER DATABASE MOUNT PARALLEL.

NOTA *L'utilizzo di CREATE DATABASE su un database esistente ne causa la distruzione.*

CREATE DATABASE LINK

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE SYNONYM, SELECT, Capitolo 21

SINTASSI CREATE [PUBLIC] DATABASE LINK *link*
 CONNECT TO *utente* IDENTIFIED BY *password*
 USING '*stringa_connessione*';

DESCRIZIONE *link* è il nome dato al link di database. *stringa_connessione* è la definizione del database remoto a cui è possibile accedere attraverso SQL*NET e definisce il link tra il database locale e il nome di un utente sul database remoto. I link PUBLIC possono essere creati solamente dal DBA, 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 link risulta disponibile solo all'utente che ha eseguito l'istruzione CREATE DATABASE LINK. *stringa_connessione* è il descrittore di connessione di SQL*NET per il database remoto.

L'accesso alle tabelle remote è analogo all'accesso alle tabelle locali, fatta eccezione per il fatto che il nome della tabella remota nella clausola from dell'istruzione select deve essere seguito da @*link*. Nella maggior parte dei sistemi viene impostato un numero massimo di link simultanei pari a quattro. Il DBA può incrementare questo numero tramite il parametro OPEN_LINKS di init.ora.

Le query a tre strutture sono limitate; non possono utilizzare l'operatore PRIOR fatta eccezione per la clausola connect by. START WITH non può contenere una sottoquery. CONNECT BY e START WITH non possono utilizzare la funzione USERENV('ENTRYID') o RowNum.

Per creare un link di database è necessario essere in possesso del privilegio CREATE DATABASE LINK e del privilegio CREATE SESSION nel database remoto. Per creare un link a un database pubblico, è necessario disporre del privilegio di sistema CREATE PUBLIC DATABASE LINK. Vedere il Capitolo 21 per ulteriori dettagli ed esempi riguardo ai link di database.

ESEMPI Nell'esempio che segue viene definito un collegamento di nome BOSS con George e viene assegnata la password TAL per il database EDMESTON:

```
create database link BOSS
connect to George identified by TAL
using 'EDMESTON';
```

A questo punto è possibile effettuare query sulle tabelle di George, come mostrato di seguito:

```
select DataAzione, Oggetto, Importo
from REGISTRO@BOSS;
```

È inoltre possibile creare un sinonimo per celare il fatto che le tabelle di Giorgio siano remote:

```
create synonym TALBOT_REGISTRO for REGISTRO@BOSS;
```

CREATE DIRECTORY

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE BFILE, Capitolo 27

SINTASSI CREATE [OR REPLACE] DIRECTORY *directory AS 'nome_percorso'*

DESCRIZIONE All'interno di ORACLE, una "directory" rappresenta un alias di una directory del sistema operativo. Prima di accedere ai valori del tipo di dato BFILE, è necessario creare una directory. Vedere il Capitolo 27.

CREATE FUNCTION

TIPO Comando SQL

PRODOTTI PL/SQL

VEDERE ANCHE ALTER FUNCTION; BLOCCO, STRUTTURA; CREATE LIBRARY; CREATE PACKAGE; CREATE PROCEDURE; DATI, TIPI; DROP FUNCTION; Capitoli 22 e 24

SINTASSI CREATE [OR REPLACE] FUNCTION [*utente.*]*funzione*
[*(argomento*[IN|OUT|IN OUT] *tipodato*[, *parametro* [IN|OUT|IN
OUT] *tipodato*...]]
RETURN *tipodato* {IS | AS} *blocco*

DESCRIZIONE *funzione* rappresenta il nome della funzione da definire. Una funzione può avere parametri, ovvero argomenti di un certo tipo a cui è stato assegnato un nome, e ciascuna funzione restituisce un valore di un certo tipo come specificato

dalla clausola di restituzione. Il blocco PL/SQL definisce il funzionamento 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 opzionale. 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 essere in possesso 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**.

In ORACLE8, la funzione dell'utente può utilizzare le librerie del C memorizzate all'esterno del database. Vedere **CREATE LIBRARY**.

CREATE INDEX

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE **ANALYZE**, **ALTER INDEX**, **DROP INDEX**, **VINCOLO DI INTEGRITÀ**, **STORAGE**, Capitoli 17 e 19

SINTASSI **CREATE** [UNIQUE | BITMAP] INDEX [*utente.*] *indice*
 ON { [*utente.*] *tabella* (*colonna* [ASC | DESC]
 [, *colonna* [ASC | DESC]] ...)
 | CLUSTER [*utente.*] *cluster*}
 [*clausola_attributi_fisici*
 | {LOGGING | NOLOGGING}
 | {TABLESPACE *tablespace* | DEFAULT}
 | {NOSORT | REVERSE}] ...
 [GLOBAL PARTITION BY RANGE (*elenco_colonne*)
 (PARTITION [*nome_partizione*]
 VALUES LESS THAN (*elenco_valori*))
 [*clausola_attributi_fisici*
 | {LOGGING | NOLOGGING}], ...)
 | LOCAL [(PARTITION [*nome_partizione*]
 [*clausola_attributi_fisici*
 | {LOGGING | NOLOGGING}], ...)]]
 [PARALLEL *clausola_parallelia*]
 clausola_attributi_fisici ::=
 [PCTFREE *intero*
 | PCTUSED *intero*
 | INITTRANS *intero*
 | MAXTRANS *intero*
 | STORAGE *registra*]

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* 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 uti-

lizzare il vincolo unico sulle colonne. Se si specificano più colonne viene creato un indice composto. ASC e DESC significano rispettivamente ascendente e discendente e vengono ammessi per questioni di compatibilità con DB2, ma non hanno alcun effetto. CLUSTER è il nome della chiave del cluster che risulta indicizzata per un cluster. I cluster devono avere le loro chiavi indicizzate perché si possa accedere alle tabelle a essi associate (*vedere CREATE TABLE* per una descrizione di INITTRANS e MAXTRANS). Il valore di default 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. Se si crea un indice UNIQUE, questa clausola viene saltata.

TABLESPACE è il nome del tablespace a cui viene assegnato questo indice. STORAGE contiene sottoclausole, ciascuna descritta alla voce STORAGE. NOSORT è un'opzione il cui valore primario consiste nel ridurre il tempo di creazione di un indice, se, e solo se, i valori nella colonna da indicizzare sono già in ordine crescente. Non viene compromesso nulla se successivamente l'ordine crescente decade, ma NOSORT funziona solo se essi sono in ordine quando viene eseguito INDEX.

Se le righe non sono in ordine, INDEX restituisce un messaggio d'errore; non viene danneggiato nulla, ma viene consentito di rieseguire il comando senza l'opzione NOSORT.

PARALLEL, assieme a DEGREE e INSTANCES, specifica le caratteristiche parallele dell'indice. DEGREE specifica il numero di server per le query da utilizzare; INSTANCES specifica il modo in cui l'indice deve essere suddiviso tra le istanze di un server parallelo per l'elaborazione delle query in parallelo. Un intero *n* specifica che l'indice deve essere suddiviso tra questo numero di istanze disponibili.

Per poter creare un indice, è necessario o possedere la tabella indicizzata o disporre del privilegio INDEX sulla tabella oppure del privilegio di sistema CREATE ANY INDEX.

BITMAP crea un indice bitmap, che può essere utile per colonne con pochi valori distinti. Le clausole PARTITION creano gli indici sulle tabelle suddivise in partizioni. *Vedere* i Capitoli 17 e 19.

CREATE LIBRARY

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE FUNCTION, CREATE PACKAGE BODY, CREATE PROCEDURE, Capitolo 24

SINTASSI CREATE [OR REPLACE] LIBRARY [*utente.*]*nome_lib* {IS|AS} '*filespec*'

DESCRIZIONE CREATE LIBRARY crea un oggetto di librerie, consentendo di fare riferimento a una libreria condivisa del sistema operativo, da cui 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.

CREATE PACKAGE

TIPO Comando SQL

PRODOTTI PL/SQL

VEDERE ANCHE ALTER PACKAGE; CREATE FUNCTION; CREATE PACKAGE BODY; CREATE PROCEDURE; CURSOR; DROP PACKAGE; ECCEZIONE; RECORD; TABELLA; VARIABILI, DICHIARAZIONE; Capitolo 24

SINTASSI CREATE [OR REPLACE] PACKAGE [*utente.*] *package*
{IS | AS} *pl/sql_package_spec*

DESCRIZIONE CREATE PACKAGE imposta le specifiche per un package PL/SQL, ovvero un gruppo di procedure pubbliche, funzioni, eccezioni, variabili, costanti e cursori. Aggiungendo OR REPLACE vengono sostituite le specifiche del package se già esistono; questa operazione invalida il package e richiede la ricompilazione del corpo di quest'ultimo e di qualsiasi oggetto dipendente dalle specifiche che si sono modificate.

Inserendo all'interno del package le proprie procedure e funzioni si consente ad esse di condividere dati attraverso variabili, costanti e cursori. In questo modo, l'utente ha la possibilità di accedere all'intera collezione in una volta sola come parte di un ruolo. ORACLE accede agli elementi di un package in modo più efficiente che non se fossero separati. Se si modifica il corpo del package, che ORACLE memorizza separatamente, non è necessario ricompilare nessun elemento che utilizzi il package.

Per poter creare un package, è necessario 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.

CREATE PACKAGE BODY

TIPO Comando SQL

PRODOTTI PL/SQL

VEDERE ANCHE ALTER PACKAGE; CREATE FUNCTION; CREATE LIBRARY; CREATE PACKAGE; CREATE PROCEDURE; CURSOR; DROP PACKAGE; ECCEZIONE; RECORD; TABELLA; VARIABILI, DICHIARAZIONE; Capitoli 22 e 24

SINTASSI CREATE [OR REPLACE] PACKAGE BODY [*utente.*] *package*
{IS | AS} *corpo_package_pl_sql*

DESCRIZIONE CREATE PACKAGE BODY costruisce il corpo di un package precedentemente specificato tramite 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 PROCEDURE

TIPO Comando SQL

PRODOTTI PL/SQL

VEDERE ANCHE ALTER PROCEDURE; BLOCCO, STRUTTURA; CREATE LIBRARY; CREATE FUNCTION; CREATE PACKAGE; DATI, TIPI; DROP PROCEDURE; Capitoli 22 e 24

SINTASSI CREATE [OR REPLACE] PROCEDURE [*utente.*]*procedura*
[(*argomento*[IN | OUT | IN OUT]) *tipodato*

[, *argomento* [IN | OUT | IN OUT] *tipodato*]...)]
 {IS | AS} *blocco*

DESCRIZIONE CREATE PROCEDURE crea le specifiche e il corpo di una procedura. Una procedura può avere parametri, ovvero argomenti specificati di un certo tipo. Il blocco PL/SQL definisce il funzionamento della funzione come una serie di dichiarazioni, istruzioni del programma PL/SQL e eccezioni.

Il qualificatore IN sta a significare 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 in sé sia il significato di IN che quello di OUT; l'utente specifica un valore e la procedura lo sostituisce con un altro. Se non viene specificato alcun qualificatore, il default è IN. La differenza tra una funzione e una procedura è che la funzione restituisce un valore all'ambiente che l'ha chiamata.

In ORACLE8, il blocco PL/SQL può fare riferimento a programmi in una libreria C esterna. Vedere CREATE LIBRARY.

CREATE PROFILE

TIPO Comando SQL

PRODOTTI ALL

VEDERE ANCHE ALTER PROFILE, ALTER RESOURCE COST, ALTER SYSTEM, ALTER USER, CREATE USER, DROP PROFILE, Capitolo 18

SINTASSI CREATE PROFILE *profilo* LIMIT
 { SESSION_PER_USER
 | CPU_PER_SESSION
 | CPU_PER_CALL
 | CONNECT_TIME
 | IDLE_TIME
 | LOGICAL_READS_PER_SESSION
 | LOGICAL_READS_PER_CALL
 | COMPOSITE_LIMIT}
 { *intero* | UNLIMITED | DEFAULT }
 | { PRIVATE_SGA { *intero* [K | M]
 | UNLIMITED
 | DEFAULT } }
 | FAILED_LOGIN_ATTEMPTS
 | PASSWORD_LIFETIME
 | { PASSWORD_REUSE_TIME
 | PASSWORD_REUSE_MAX }
 | ACCOUNT_LOCK_TIME
 | PASSWORD_GRACE_TIME }
 { *intero* | UNLIMITED | DEFAULT }
 | PASSWORD_VERIFY_FUNCTION
 { *funzione* | NULL | DEFAULT } }

DESCRIZIONE CREATE PROFILE crea un insieme di limiti sull'utilizzo delle risorse di un database. Quando si associa il profilo ad un utente tramite CREATE o ALTER USER, è possibile controllare, per mezzo di questi limiti, cosa fa l'utente. Per utilizzare CREATE PROFILE, è necessario abilitare i limiti delle risorse attraverso il parametro di inizializzazione RESOURCE_LIMIT oppure il comando ALTER SYSTEM.

SESSIONS_PER_USER limita il numero di sessioni SQL contemporanee 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 dopo che è trascorso il numero di minuti specificato; ciò non si applica in fase di 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 nella SGA come privato; 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 vi è limite a una particolare risorsa; DEFAULT preleva il limite dal profilo DEFAULT, che si può modificare attraverso 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 poter 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.

Vedere il Capitolo 18 per ulteriori dettagli sugli attributi del profilo

CREATE ROLE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER ROLE, DROP ROLE, GRANT, REVOKE, ROLE, SET ROLE, Capitolo 18

SINTASSI CREATE ROLE *ruolo*

[[NOT IDENTIFIED | IDENTIFIED] [BY *password* | EXTERNALLY | GLOBALLY]]

DESCRIZIONE Con CREATE ROLE è possibile creare un ruolo con un certo nome e un insieme di privilegi. Quando si attribuisce un ruolo a un utente, vengono assegnati a quest'ultimo tutti i privilegi di quel ruolo. Per prima cosa, è necessario creare il ruolo con CREATE ROLE, successivamente concedere i privilegi al ruolo utilizzando il comando GRANT. Quando un utente vuole accedere a qualche operazione consentita dal ruolo che possiede, abilita il ruolo con SET ROLE.

Se si protegge il ruolo tramite una password, l'utente che vuole disporre dei privilegi deve fornire la password con il comando SET ROLE utilizzato per abilitare il ruolo.

ORACLE crea automaticamente diversi ruoli, inclusi CONNECT, RESOURCE, DBA, EXP_FULL_DATABASE e IMP_FULL_DATABASE. I primi tre ruoli forniscono compatibilità con le versioni precedenti di ORACLE. Gli ultimi due ruoli consentono di utilizzare le utilità di importazione ed esportazione. Vedere ROLE per un elenco completo e il Capitolo 18 per degli esempi al riguardo.

CREATE ROLLBACK SEGMENT

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER ROLLBACK SEGMENT, CREATE DATABASE, CREATE TABLESPACE, DROP ROLLBACK SEGMENT, STORAGE

SINTASSI CREATE [PUBLIC] ROLLBACK SEGMENT *segmento*
 [TABLESPACE *tablespace*]
 [STORAGE *registra*]

DESCRIZIONE *segmento* è il nome assegnato a questo segmento di rollback. *tablespace* è il nome del tablespace a cui questo segmento di rollback è assegnato. Un tablespace può avere più segmenti di rollback. Per utilizzare questo comando, è necessario essere in linea.

STORAGE contiene sottoclausole, ognuna delle quali è descritta alla voce STORAGE. Se si utilizza PUBLIC, questo segmento di rollback può essere utilizzato da qualsiasi istanza che lo richieda; in caso contrario, è disponibile solo per le istanze che lo citano esplicitamente nei rispettivi file init.ora.

CREATE SCHEMA

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE TABLE, CREATE VIEW, GRANT

SINTASSI CREATE SCHEMA AUTHORIZATION *schema*
 {CREATE TABLE *comando* |
 CREATE VIEW *comando* |
 GRANT *comando*}
 [{CREATE TABLE *comando* |
 CREATE VIEW *comando* |
 GRANT *comando*}]...}

DESCRIZIONE Il comando CREATE SCHEMA crea un insieme di tabelle, viste e concessioni di privilegi come un'unica transazione. Il nome dello schema è identico al nome dell'utente di ORACLE. I comandi CREATE TABLE, CREATE VIEW e GRANT sono i comandi standard e l'ordine con cui appaiono i comandi non è importante, anche se vi sono dipendenze interne.

CREATE SEQUENCE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER SEQUENCE, AUDIT, DROP SEQUENCE, GRANT, REVOKE, NEXTVAL e CURRVAL alla voce PSEUDOCOLONNE, Capitolo 19

SINTASSI CREATE SEQUENCE [*utente.*]*sequenza*
 [INCREMENT BY *intero*]
 [START WITH *intero*]
 [MAXVALUE *intero* | NOMAXVALUE]
 [MINVALUE *intero* | NOMINVALUE]
 [CYCLE | NOCYCLE]
 [CACHE *intero* | NOCACHE]
 [ORDER | NOORDER]

DESCRIZIONE *sequenza* è il nome dato alla sequenza. Il default di INCREMENT BY è 1. Un numero positivo causa un incremento in ordine crescente del numero della sequenza in base al valore dell'intero. Un numero negativo causa la stessa cosa, ma in ordine decrescente. START WITH è il numero con cui la sequenza ha inizio.

Il valore di default per START WITH è MAXVALUE per le sequenze in ordine decrescente e MINVALUE per quelle in ordine crescente; si utilizza START WITH per riassegnare questo valore di default.

MINVALUE è il numero più basso che la sequenza può generare. Il valore di default è 1 per le sequenze in ordine crescente. MAXVALUE è il numero più alto che la sequenza può generare. Per consentire alle sequenze di progredire senza limiti, è sufficiente specificare solo MINVALUE per le sequenze in ordine crescente e MAXVALUE per quelle in ordine decrescente. Per interrompere la creazione di numeri della sequenza e forzare un errore nel caso venga fatto un tentativo di ottenere un nuovo numero, è necessario 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 sequenza dal punto corrispondente al relativo MAXVALUE o MINVALUE, occorre specificare CYCLE.

CACHE consente di tenere in memoria un gruppo preallocato di numeri di una sequenza. Il valore di default è pari a 20. Il valore impostato deve essere minore di MAXVALUE meno MINVALUE.

ORDER garantisce che i numeri della sequenza vengano assegnati alle istanze che li richiedono nell'ordine con cui vengono ricevute le richieste. Ciò risulta particolarmente utile nelle applicazioni che richiedono un resoconto della sequenza in cui hanno avuto luogo le transazioni.

Per creare una sequenza all'interno del 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

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER SNAPSHOT, CREATE SNAPSHOT LOG, DROP SNAPSHOT, STORAGE, Capitolo 28

SINTASSI CREATE SNAPSHOT [*utente.*] *snapshot*

```

    { { PCTFREE intero
        | PCTUSED intero
        | INITRANS intero
        | MAXTRANS intero
        | TABLESPACE tablespace
        | STORAGE registra_ }
    | CLUSTER cluster (colonna [, colonna] ...) } ...
    [USING {INDEX [ PCTFREE intero
                    | PCTUSED intero
                    | INITRANS intero
                    | MAXTRANS intero ]
             | [ DEFAULT ] [ MASTER | LOCAL ] ROLLBACK SEGMENT
               [ segmento rollback ] } ]
    [REFRESH [FAST | COMPLETE | FORCE] [START WITH data] [NEXT date]
              [WITH {PRIMARY KEY | ROWID}]] ]
    [FOR UPDATE] AS sottoquery
  
```

DESCRIZIONE CREATE SNAPSHOT crea uno snapshot, ovvero una tabella che contiene i risultati di una query, solitamente su una o più tabelle (chiamate tabelle master) in un database remoto.

Questo consente di avere una copia locale di sola lettura dei dati per incrementare la velocità di esecuzione delle query sui dati. Si possono riaggiornare i dati a intervalli di tempo utilizzando la clausola REFRESH. Gli snapshot non possono contenere colonne di tipo LONG.

Il primo gruppo di opzioni imposta la registrazione della tabella locale. In alternativa, si può impostare lo snapshot in un cluster tramite la clausola CLUSTER. La clausola USING INDEX consente di specificare i parametri di memorizzazione per l'indice dello snapshot.

Un aggiornamento veloce (FAST) utilizza il log dello snapshot associato alla tabella master per aggiornare lo snapshot. Un aggiornamento completo (COMPLETE) riesegue la query. Un aggiornamento forzato (FORCE) comunica a ORACLE di effettuare una scelta tra l'opzione FAST e l'opzione COMPLETE. ORACLE per prima cosa aggiorna lo snapshot alla *data* indicata tramite START WITH. Se viene fornita una *data* successiva (NEXT), ORACLE aggiorna lo snapshot a intervalli di tempo determinati dalla differenza tra la data START WITH e NEXT.

Uno snapshot semplice seleziona i dati da un'unica tabella master utilizzando una query semplice. Uno snapshot complesso seleziona i dati utilizzando una sotto-query GROUP BY o CONNECT BY, un'unione oppure operazioni di assegnamento nella query. ORACLE può eseguire un aggiornamento veloce FAST solo su snapshot semplici che dispongono di log di snapshot.

A causa degli oggetti locali creati dagli snapshot, il nome di uno snapshot non può eccedere i 19 caratteri di lunghezza. Per creare uno snapshot nel proprio schema, è necessario disporre del privilegio di sistema CREATE SNAPSHOT. Per creare uno snapshot nello schema di un altro utente, occorre possedere il privilegio di sistema CREATE ANY SNAPSHOT. Vedere il Capitolo 28.

CREATE SNAPSHOT LOG

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER SNAPSHOT LOG, CREATE SNAPSHOT, DROP SNAPSHOT LOG, STORAGE, Capitolo 28

SINTASSI CREATE SNAPSHOT LOG ON [*utente.*] *tabella*
 [WITH {[PRIMARY KEY] [,ROWID] [,,(*filtro colonna*)
 | ,(*filtro colonna*) ...]}
 [PCTFREE *intero* |
 PCTUSED *intero* |
 INITRANS *intero* |
 MAXTRANS *intero* |
 TABLESPACE *tablespace* |
 STORAGE *registra*]]

DESCRIZIONE CREATE SNAPSHOT LOG crea una tabella, associata con la tabella master di uno snapshot che tiene traccia delle modifiche effettuate ai dati della tabella master. ORACLE utilizza il log di snapshot per aggiornare velocemente (FAST) gli snapshot di una tabella master. Le opzioni di memorizzazione specificano la registrazione della tabella. ORACLE registra le modifiche solo se c'è uno snapshot semplice che si basa su una tabella master. Si può avere solo un log per una determinata tabella master.

A causa degli oggetti locali che vengono creati come supporto al log di snapshot, il nome della tabella di base su cui è costituito lo snapshot non può superare i 24 caratteri di lunghezza.

Per creare un log di snapshot è necessario possedere i privilegi di sistema CREATE TABLE, CREATE TRIGGER e CREATE INDEX.

CREATE SYNONYM

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE DATABASE LINK, CREATE TABLE, CREATE VIEW, Capitoli 18 e 21

SINTASSI CREATE [PUBLIC] SYNONYM [*utente.*]*sinonimo*
FOR [*utente.*]*tabella* [@*link_database*];

DESCRIZIONE CREATE SYNONYM crea un sinonimo per il nome di una tabella o una vista, incluse quelle su un database remoto. Esegue il commit delle modifiche pendenti nel database.

PUBLIC rende disponibile il sinonimo a tutti gli utenti, ma può essere creato così solo dal 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 a cui si fa riferimento. Si può creare il sinonimo di un sinonimo. @*link_database* è il riferimento del link a un database remoto. Il sinonimo si riferisce a una tabella nel database remoto come specificato dal link di database.

ESEMPIO create synonym TALBOT_REGISTRO for REGISTRO@BOSS;

CREATE TABLE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER TABLE; CREATE CLUSTER; CREATE INDEX; CREATE TABLESPACE; DATI, TIPI; DROP TABLE; VINCOLO DI INTEGRITÀ; NOMI DEGLI OGGETTI; STORAGE; Capitoli 17, 19, 27 e 31

SINTASSI Definizione di tabelle relazionali:=

```
CREATE TABLE [utente.] tabella
[({ colonna tipodato [DEFAULT espr] [WITH ROWID]
      [SCOPE IS [utente.] nome_tabella_ambito]
          [limitazione_colonna] ...
      | limitazione_tabella | REF (rif_nome_colonna) WITH ROWID
      | SCOPE FOR (rif_nome_colonna) IS
          [utente.]nome_tabella_ambito }
  [, {colonna tipodato [DEFAULT espr] [WITH ROWID]
      [SCOPE IS [utente.] nome_tabella_ambito]
          [limitazione_colonna] ...
      | limitazione_tabella | REF (rif_nome_colonna) WITH ROWID
      | SCOPE FOR (rif_nome_colonna) IS
          [utente.]nome_tabella_ambito} ] ...) ]
  [ {[ORGANIZATION {HEAP | INDEX}
      | PCTTHRESHOLD [INCLUDING nome_colonna]
```

```

[OVERFLOW [clausola_attributi_fisici
           | TABLESPACE tablespace] ...]
| clausola_attributi_fisici
| TABLESPACE tablespace
| LOB (lob_elem [, lob_elem ...]) STORE AS
[lob_nomeseg]
[ (TABLESPACE tablespace
  | STORAGE registra
  | CHUNK intero
  | PCTVERSION intero
  | CACHE
  | NOCACHE LOGGING | NOCACHE NOLOGGING
  | INDEX [lob_nome_indice]
  [ ( TABLESPACE tablespace
    | STORAGE registra
    | INITTRANS intero
    | MAXTRANS intero ) ...])
| NESTED TABLE elem_annidato STORE AS tabella_registra
| {LOGGING | NOLOGGING} ...
| CLUSTER cluster (colonna [, colonna] ...)]
| [PARALLEL clausola_parallelia]
| [PARTITION BY RANGE (elenco_colonne)
  (PARTITION [nome_partizione] VALUES LESS THAN (elenco_valori)
   [clausola_attributi_fisici
   | TABLESPACE tablespace
   | {LOGGING | NOLOGGING} ...
  [ ENABLE clausola_abilita | DISABLE clausola_disabilita] ...
  [AS sottoquery]
  [CACHE | NOCACHE]

```

`clausola_attributi_fisici ::=`

```

[ PCTFREE intero
| PCTUSED intero
| INITTRANS intero
| MAXTRANS intero
| STORAGE registra_ ]

```

`Definizione di tavelli di oggetti::=`

```

CREATE TABLE [utente.]tabella OF [utente.]tipo Oggetto
[ ( [colonna | attributo [DEFAULT espr] [WITH ROWID]
      [SCOPE IS [utente.]
       nome_tabella_ambito] [limitazione_colonna] ...]
     | limitazione_tabella | REF (rif_nome_colonna) WITH ROWID
     | SCOPE FOR (rif_nome_colonna) IS
       [utente.]nome_tabella_ambito )
  [, {colonna | attributo [DEFAULT espr] [WITH ROWID]
        [SCOPE IS [utente.] nome_tabella_ambito]
        [limitazione_colonna] ...}

```

```

| limitazione_tabella | REF (rif_nome_colonna) WITH ROWID
| SCOPE FOR (rif_nome_colonna) IS
|           [utente.]nome_tabella_ambito] ...) ]
|OIDINDEX [indice] [(clausola_attributi_fisici
|           TABLESPACE tablespace) ...]
[ { [clausola_attributi_fisici
| TABLESPACE tablespace
| LOB (lob_elem [, lob_elem...]) STORE AS
|       [lob_nomeseg]
|       [( TABLESPACE tablespace
|           STORAGE registra
|           CHUNK intero

| PCTVERSION intero
| CACHE
| NOCACHE LOGGING | NOCACHE NOLOGGING
| INDEX [lob_nome_indice]
|       [( TABLESPACE tablespace
|           STORAGE registra
|           INITTRANS intero
|           MAXTRANS intero )] )
| NESTED TABLE elem_annidato STORE AS tabella_registra
|       {LOGGING | NOLOGGING} ] ...
| CLUSTER cluster (colonna[, colonna] ...)]
|PARALLEL clausola_parallelia]
[ ENABLE clausola_abilita
| DISABLE clausola_disabilita] ...
[AS sottoquery]

[CACHE | NOCACHE]
```

DESCRIZIONE *utente* rappresenta il proprietario della tabella. Se viene omesso, si considera come proprietario della tabella, l'utente che ha eseguito il comando. *tabella* è il nome della tabella e segue le convenzioni di nomenclatura di ORACLE.

colonna è il nome di una colonna e *tipodato* può essere CHAR, VARCHAR, VARCHAR2, DATE, LONG, NUMBER, ROWID, MLSLABEL, RAW MLSLABEL, RAW, BLOB, CLOB, NCLOB, BFILE o LONG RAW (vedere DATI, TIPI). DEFAULT specifica un valore da assegnare alla colonna se viene inserita una riga che non possiede un valore per quella 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.

Vedere VINCOLO DI INTEGRITÀ per una descrizione completa i vincoli sulle tabelle e sulle colonne.

CLUSTER include la tabella in questione nel cluster specificato. Le colonne della tabella elencate devono corrispondere, nell'ordine e nel tipo, 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 si sta facendo.

INITTRANS comunica il numero iniziale di transazioni che possono aggiornare contemporaneamente un blocco di dati (le operazioni di select non vengono

contate). La sequenza di valori per INITRANS va da 1 a 255. 1 è il valore di default. Ciascuna transazione occupa spazio (23 byte nella maggior parte dei sistemi) nel blocco di dati stesso fino a che essa non risulta completata. Quando viene creato per un blocco un numero di transazioni maggiore di INITRANS, lo spazio per queste transazioni viene automaticamente allocato, fino a un massimo pari a MAXTRANS.

MAXTRANS comunica il numero massimo di transazioni che possono aggiornare contemporaneamente un blocco di dati (le operazioni di select non vengono contate). MAXTRANS va da 1 a 255. 255 è il valore di default. Ciascuna transazione occupa spazio (23 byte sulla maggior parte dei sistemi) nel blocco di dati stesso fino a che essa non risulta completata. Le code di transazioni in attesa di essere eseguite occupano sempre più spazio all'interno del blocco, nonostante solitamente vi sia più spazio libero di quello necessario per tutte le transazioni contemporanee.

Ogni volta che ORACLE inserisce una riga in una tabella, prima controlla per vedere quanto spazio risulta disponibile nel blocco corrente (la dimensione di un blocco dipende dal sistema operativo, vedere *ORACLE Installation and User's Guide* del proprio sistema). Se la dimensione della riga lascia meno della percentuale PCTFREE nel blocco, la riga viene inserita in un nuovo blocco. Il default è 10; 0 è il valore minimo.

PCTUSED ha un valore di default pari a 40. Questo rappresenta la percentuale minima di spazio disponibile in un blocco perché quest'ultimo rappresenti un candidato per l'inserimento di nuove righe. ORACLE tiene traccia di quanto spazio in un blocco è stato liberato dalle cancellazioni. Se risulta al di sotto di PCTUSED, ORACLE lo rende disponibile per i nuovi inserimenti.

STORAGE contiene sottoclausole che vengono descritte sotto **STORAGE**. **TABLESPACE** è il nome del tablespace a qui è assegnata la tabella in questione.

Le clausole enable e disable abilitano o disabilitano le limitazioni. Vedere **ENABLE** e **DISABLE**.

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 suddividere la tabella tra le istanze di un server parallelo per l'elaborazione di query in parallelo. Un intero *n* indica che la tabella deve essere suddivisa tra il numero specificato di istanze disponibili.

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 con dati di tipo **LONG**.

Durante l'operazione **create table... as select**, si possono disabilitare le operazioni di log, tramite la parola chiave **NOLOGGING**. Questa opzione disabilita le voci redo log che vengono normalmente scritte durante il riempimento della nuova tabella, migliorando così le prestazioni, ma compromettendo la possibilità da parte dell'utente di ripristinare quei dati nel caso si verificasse un fallimento dell'istanza prima del successivo salvataggio.

La clausola **LOB** specifica la memorizzazione dei dati fuori linea per dati **LOB** (**BLOB**, **CLOB** e **NCLOB**) registrati internamente. Vedere il Capitolo 27.

La sezione "Definizione a oggetti delle tabelle" si applica alle tabelle di oggetti in cui ciascuna riga contiene l'ID di un oggetto (OID). Vedere il Capitolo 31.

CREATE TABLESPACE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER TABLESPACE, DROP TABLESPACE, Capitolo 19

SINTASSI CREATE TABLESPACE *tablespace*

```
DATAFILE definizione_file [,definizione_file]...
  [AUTOEXTEND definizione_file [definizione_file] [ON | OFF]
    [NEXT intero [K | M]]
    [MAXSIZ [UNLIMITED| intero [K | M]]]
    [MINIMUM EXTENT intero [K | M] ]
    [DEFAULT STORAGE registra]
    [ONLINE | OFFLINE]
    [PERMANENT | TEMPORARY ]
```

DESCRIZIONE *tablespace* è il nome di un tablespace e segue le regole di nomenclatura di ORACLE. DATAFILE è un file o una serie di file descritti in base a *definizione_file*, in cui vengono specificati i nomi e le dimensioni dei file del database:

```
'file' [SIZE intero [K | M] [REUSE]
```

Il formato del file dipende dal sistema operativo. SIZE è 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 invece per 1048576. DEFAULT STORAGE definisce lo spazio di default in cui memorizzare tutti gli oggetti creati nel tablespace, a meno che i valori di default non vengano riscritti, ad esempio, tramite CREATE TABLE. ONLINE, il valore di default, indica che questo tablespace risulterà disponibile agli utenti subito dopo la sua creazione. OFFLINE impedisce l'accesso al tablespace fino a che il comando ALTER TABLESPACE non lo modifica in ONLINE. DBA_TABLESPACES fornisce lo stato di tutti i tablespace.

SIZE e REUSE assieme comunicano a ORACLE di riutilizzare il file se già esiste (il suo contenuto viene distrutto) oppure crearlo se non esiste ancora. SIZE senza REUSE crea un file che non esiste ancora e restituisce un errore nel caso in cui il file esista già.

Il parametro MINIMUM EXTENT consente di specificare la dimensione minima per le estensioni all'interno del tablespace. Il valore di default è specifico per ciascun sistema operativo e dipende dalla dimensione dei blocchi del database.

Se si abilita (ON) l'opzione AUTOEXTEND, un file di dati viene esteso in modo dinamico con incrementi di dimensione pari a NEXT, fino a un massimo di MAXSIZE (o UNLIMITED, illimitato). Se un tablespace viene utilizzato solo per segmenti temporanei creati durante l'elaborazione della query, è possibile crearlo come un tablespace di tipo TEMPORARY tablespace. Se, invece, è destinato a contenere oggetti permanenti (come le tabelle), è consigliabile utilizzare l'impostazione di default pari a PERMANENT.

CREATE TRIGGER

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER TABLE; ALTER TRIGGER; BLOCCO, STRUTTURA; DROP TRIGGER; Capitoli 23 e 25

SINTASSI CREATE [OR REPLACE] TRIGGER [*utente.*]*trigger*
 {BEFORE | AFTER | INSTEAD OF}
 {DELETE | INSERT | UPDATE [OF *colonna*[, *colonna*]...}
 [OR {DELETE | INSERT | UPDATE [OF *colonna*[, *colonna*]...}]}...
 ON [*utente.*]*tabella*
 [REFERENCING {OLD [AS] *vecchio* | NEW [AS] *nuovo*}]
 [FOR EACH ROW]
 [WHEN (*condizione*)]
 blocco

DESCRIZIONE CREATE TRIGGER crea e abilita un trigger per un database, ovvero una procedura memorizzata associata a una tabella, indicata nella clausola on, che ORACLE esegue automaticamente quando l'istruzione SQL specificata viene eseguita sulla tabella in questione. I trigger possono essere utilizzati per forzare limitazioni complesse e per propagare le modifiche in tutto il database anziché eseguire questa operazione nelle proprie applicazioni. In questo modo, è sufficiente implementare il codice per il trigger una volta sola anziché doverlo ripetere per ogni applicazione.

Per creare un trigger è necessario possedere il privilegio CREATE TRIGGER o CREATE ANY TRIGGER e l'utente deve poter disporre direttamente di tutti i privilegi necessari a eseguire la procedura memorizzata.

La clausola principale del comando CREATE TRIGGER specifica l'operazione SQL che innesca il blocco (delete, insert, update) e il momento in cui il trigger scatta, prima, dopo o al posto (BEFORE, AFTER o INSTEAD OF) dell'esecuzione dell'operazione di trigger.

Per un trigger di tipo UPDATE è necessario specificare una clausola OF, in modo che il trigger scatti solo quando vengono aggiornate le colonne specificate.

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 di default sono OLD e NEW.

La clausola FOR EACH ROW specifica un trigger di riga, il trigger scatta ogni volta che ciascuna riga viene coinvolta in una operazione associata a trigger. La clausola WHEN limita l'esecuzione del trigger in modo che avvenga solo quando incontra una certa *condizione*. Si tratta di una condizione SQL, non PL/SQL.

I comandi ALTER TRIGGER e ALTER TABLE consentono di disabilitare e abilitare i trigger. Se un trigger è disabilitato, 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.

Vedere i Capitoli 23 e 25 per esempi in proposito.

CREATE TYPE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE TABLE, CREATE TYPE BODY, Capitoli 24, 25 e 26

SINTASSI *crea_tipo_varray ::=*

```
CREATE TYPE nome_tipo AS {VARRAY | VARYING ARRAY} (limit) OF
tipodato;
```

crea_tipo_tabella_annidata ::=

```
CREATE TYPE nome_tipo AS TABLE OF tipodato;
```

crea_spec_tipo Oggetto ::=

```
CREATE [OR REPLACE] TYPE [utente.]nome_tipo AS OBJECT
(
    nome_attributo tipodato[, nome_attributo tipodato]...
    | [{MAP | ORDER} MEMBER specifiche_funzione]
    | [MEMBER {specifiche_procedura | specifiche_funzione}...]
    [, MEMBER {specifiche_procedura | specifiche_funzione}...]
    | [PRAGMA RESTRICT_REFERENCES (nome_metodo, limitazioni)
    [, PRAGMA RESTRICT_REFERENCES (nome_metodo, limitazioni)]...]
);
limitazioni ::=
{RNDS | WNDS | RNPS | WNPS}
```

crea_tipo Oggetto incompleto ::=

```
CREATE [OR REPLACE] TYPE [utente.]nome_tipo AS OBJECT;
```

DESCRIZIONE CREATE TYPE consente di creare un tipo di dato astratto, di nome 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, è necessario disporre dei privilegi di sistema CREATE ANY TYPE.

Durante la creazione di un tipo di dato astratto, ci si può basare sui tipi di dati forniti da ORACLE (quali DATE e NUMBER) e sugli eventuali tipi di dati astratti definiti in precedenza.

Un tipo “incompleto” possiede un nome ma non ha attributi o metodi. Tuttavia, ad esso possono fare riferimento altri tipi di oggetti e, quindi, può essere utilizzato per definire tipi di oggetti che fanno riferimento l’uno all’altro.

Se si intende creare dei metodi per il tipo in questione, è necessario dichiarare i nomi dei metodi nelle specifiche 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:

- WNDS Write No Database State (non modifica alcuna tabella);
- WNPS Write No Package State (non modifica le variabili del package);
- RNDS Read No Database State (non esegue query);
- RNPS Read No Package State (non fa riferimento a variabili del package).

Come minimo, gli utenti dovrebbero utilizzare la limitazione WNDS sui propri metodi. Vedere il Capitolo 25. Per ulteriori informazioni sui tipi VARRAY e le tabelle annidate vedere il Capitolo 26.

ESEMPI Creazione di un tipo con associati dei metodi:

```
create or replace type ANIMAL_TI as object
```

```
(Razza      VARCHAR2(25),
Nome       VARCHAR2(25),
DataNasc   DATE,
member function ETÀ (DataNasc IN DATE) return NUMBER,
PRAGMA RESTRICT_REFERENCES(ETÀ, WNDS));
```

Creazione di un varray:

```
create or replace type STRUM_VA as varray(5) of VARCHAR2(25);
```

Creazione di un tipo senza metodi:

```
create type INDIRIZ_TY as object
```

```
(Via      VARCHAR2(50),
Citta    VARCHAR2(25),
Stato    CHAR(2),
CAP      NUMBER);
```

Creazione di un tipo che utilizza un altro tipo:

```
create type PERSONA_TY as object
(Nome      VARCHAR2(25),
Indiriz   INDIRIZ_TY);
```

CREATE TYPE BODY

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE FUNCTION, CREATE PROCEDURE, CREATE TYPE, Capitoli 4, 22, 24, 25 e 26

SINTASSI CREATE [OR REPLACE] TYPE BODY [utente.]*nome_tipo*
IS
MEMBER {dichiarazione_procedura | dichiarazione_funzione};
[MEMBER {dichiarazione_procedura | dichiarazione_funzione};] ...
[{MAP | ORDER} MEMBER *dichiarazione_funzione*;]
END;

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 25 per degli esempi relativi ai metodi.

ESEMPIO create or replace type body ANIMALE_TY as
member function Eta (DataNasc DATE) return NUMBER is
begin
 RETURN ROUND(SysDate - DataNasc);
end;
end;
/

CREATE USER

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER USER, CREATE PROFILE, CREATE ROLE, CREATE TABLESPACE, GRANT, Capitolo 18

SINTASSI CREATE USER *utente* IDENTIFIED {BY *password* | EXTERNALLY}
 [DEFAULT TABLESPACE *tablespace*]
 [TEMPORARY TABLESPACE *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 gli consenta di accedere al database disponendo di un certo gruppo di privilegi e di impostazioni di memorizzazione. Se si specifica una password, è necessario fornire questa password al momento del collegamento; 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 il tablespace in cui gli utenti creano gli oggetti. TEMPORARY TABLESPACE è il tablespace in cui vengono creati gli oggetti temporanei per le operazioni dell'utente.

È possibile attribuire su entrambi i tablespace una QUOTA che limiti 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 regolare l'utilizzo delle risorse del database. Se non viene specificato un profilo, ORACLE assegna all'utente il profilo DEFAULT.

Quando viene creato un utente, inizialmente, non ha privilegi. Per attribuire ruoli e privilegi all'utente, è necessario utilizzare il comando GRANT. Normalmente, come minimo privilegio viene concesso CREATE SESSION.

CREATE VIEW

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE SYNONYM, CREATE TABLE, DROP VIEW, RENAME, VINCOLO DI INTEGRITÀ, Capitoli 17 e 25

SINTASSI CREATE VIEW [OR REPLACE] [FORCE | NO FORCE] VIEW [*utente..*] *vista*
 [[(*alias* [*alias*] ...)]
 | OF [*utente..*]*nome_tipo* WITH OBJECT OID [DEFAULT
 | (*attributo* [, *attributo*] ...)]]
 AS *sottoquery*
 [WITH [READ ONLY
 | CHECK OPTION [CONSTRAINT *vincolo*]]]

DESCRIZIONE CREATE VIEW definisce una vista. Esegue il commit di modifiche pendenti nel database di default. L'opzione OR REPLACE ricrea la vista se già esiste. L'opzione FORCE crea la vista indipendentemente dal fatto che le tabelle a cui essa fa riferimento esistano oppure no o se l'utente possiede i privilegi necessari su di esse.

L'utente non può ancora eseguire la vista, ma la può creare. L'opzione NOFORCE crea la vista solo se le tabelle di base esistono e se l'utente possiede i privilegi su di esse.

utente è il nome dell'utente per il quale la vista viene creata. *vista* è il nome assegnato alla vista in questione.

Se si specifica un alias, la vista lo utilizza come nome della colonna corrispondente all'interno della query. Se non viene specificato alcun alias, la vista eredita il nome della tabella che compare nella query; in questo caso, ciascuna colonna della query deve possedere un nome univoco, conforme alle convenzioni di nomenclatura di ORACLE: non può essere un'espressione o la colonna di una tabella. Anche un alias presente nella query stessa 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 quest'ultima determina quali righe debbano essere recuperate.

WITH CHECK OPTION limita gli inserimenti e gli aggiornamenti eseguiti attraverso la vista per evitare che vengano create righe che la vista non è in grado da sola di selezionare con la clausola where dell'istruzione CREATE VIEW. Perciò, il seguente comando:

```
create or replace view DONNE
  as select Nome, Dipartimento, Sesso
    from IMPIEGATI where Sesso = 'F'
  with check option;
```

evita di inserire una riga nella tabella DONNE che abbia il campo Sesso uguale a M o NULL, oppure di modificare il valore della colonna Sesso tramite un'operazione di update.

WITH CHECK OPTION può essere utilizzato in una vista che si basa su un'altra vista; tuttavia, se la vista sottostante possiede già un'opzione WITH CHECK OPTION, viene ignorato.

vincolo è il nome assegnato a CHECK OPTION; si tratta di un nome opzionale assegnato a questo vincolo. Senza di esso, ORACLE assegna un nome nella forma SYS_C_n, dove *n* è un intero. Un nome assegnato da ORACLE solitamente viene modificato nel corso di un'operazione di importazione, mentre un nome assegnato dall'utente non cambia.

update e delete funziona sulle righe di una vista se la vista è basata 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 è basata su un'unica tabella e se la sua query non contiene la clausola group by, la clausola distinct, funzioni di gruppo, riferimenti a pseudocolonne o qualsiasi espressione. 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 modifica dei dati.

Si possono creare viste di oggetti per sovrapporre tipi di dati astratti sulle tabelle relazionali esistenti. Vedere il Capitolo 25.

Per poter creare una vista, è necessario disporre del privilegio di sistema CREATE VIEW. Per creare una vista nello schema di un altro utente, è necessario 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'utilizzo 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.

CRT, FILE

Un file CRT è un contenitore di informazioni relative a un terminale hardware, i suoi tasti chiave e la relativa corrispondenza con i programmi di ORACLE, oltre alle sequenze di escape generali per abilitare impostazioni differenti sul terminale stesso.

CTXCTL

L'utilità CTXCTL consente di semplificare la gestione dei processi del server relativi a ConText. Permette di avviare e arrestare velocemente i server di ConText, così come di generare un semplice report dello stato dei server attivi in un dato momento. I comandi disponibili sono i seguenti.

| | |
|-------------------------------------|---|
| help [comando] | Visualizza le informazioni di aiuto per un comando specifico. |
| status | Visualizza i server attivi. |
| start n [ling query ddl dml] | Avvia <i>n</i> server. |
| stop pid... all | Arresta i processi del server. |
| quit | Arresta ctxctl. |
| exit | Esce da ctxctl. |

Vedere il Capitolo 30 per informazioni sulla configurazione di un database utilizzato da query di ConText.

CURRVAL

Vedere PSEUDOCOLONNE.

CURSOR

Cursore possiede due definizioni, riportate di seguito.

- Un cursore è un marcatore, quale un quadratino luminoso o una linea, che segnala la posizione corrente sullo schermo CRT.
- Cursore è anche un sinonimo di area di contesto, un'area di lavoro all'interno della memoria in cui ORACLE registra le istruzioni SQL correnti. Per una query, l'area in memoria include anche le intestazioni delle colonne e una riga recuperata dall'istruzione select.

CURSOR, COORDINATE

Le coordinate di un cursore sono le coordinate della posizione orizzontale e verticale di un cursore sullo schermo. Le etichette e gli intervalli di valore per le coordinate variano a seconda del tipo di terminale.

CURSOR, DESCRITTORE

Il descrittore di un cursore rappresenta un'area riservata per ciascun cursore (area di contesto) che contiene alcune informazioni sul cursore.

CURSOR, ID

L'ID di un cursore è il nome dato a ciascun cursore o area di contesto.

CURSOR, PL/SQL

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE PACKAGE, CREATE PACKAGE BODY, Capitolo 22

SINTASSI CURSOR *cursore* [(*parametro tipodato*[,*parametro tipodato*]...)]
[IS query]

DESCRIZIONE In un package PL/SQL è possibile specificare un cursore e dichiarare il suo corpo. La specifica contiene solo l'elenco dei parametri con i loro tipi di dati corrispondenti, non la clausola IS, mentre il corpo le contiene entrambe. I parametri possono comparire in qualsiasi punto della query in cui possa comparire una costante. È possibile specificare il cursore come parte delle dichiarazioni pubbliche delle specifiche del package e successivamente il corpo del cursore come parte del corpo del package nascosto.

CURSOR SQL

TIPO Cursore隐式的PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE %FOUND, %ISOPEN, %NOTFOUND, %ROWCOUNT, Capitolo 22

DESCRIZIONE *SQL* è il nome del cursore aperto implicitamente ogni volta che viene elaborata un'istruzione SQL che non lavori su un cursore esplicitamente aperto (*vedere DECLARE*). Gli attributi %FOUND e %NOTFOUND possono essere controllati verificando il valore di SQL%FOUND e SQL%NOTFOUND prima o dopo insert, update o delete (i quali non sono mai associati a cursori esplicativi) o dopo un comando select su riga singola che sia stato eseguito senza cursori esplicativi.

Vedere %FOUND per ulteriori dettagli riguardanti queste verifiche. *Vedere* %ROWCOUNT per dettagli inerenti gli effetti di questi attributi in condizioni differenti. SQL%ISOPEN vale sempre FALSE dato che ORACLE chiude il cursore SQL automaticamente al termine dell'istruzione.

DATABASE SMONTATO

Un database smontato è un database che non è stato montato da alcuna istanza; quindi, non può essere aperto e non è disponibile per l'utilizzo.

DATA CONTROL LANGUAGE (DCL), ISTRUZIONI

Le istruzioni DCL rappresentano una categoria di istruzioni SQL. Le istruzioni DCL, quali grant connect, grant select, grant update e revoke dba, 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 dal database. Esempi di questo tipo di istruzioni sono create view, create table, create index, drop table e rename table. Le altre categorie sono istruzioni del linguaggio per il controllo dei dati (DCL) o 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 due definizioni, riportate di seguito.

- 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 registra le tabelle, le viste e altri oggetti; inoltre, l'insieme di oggetti del database utilizzati da una certa applicazione.

DATABASE, LINK

Un link di database è un oggetto memorizzato nel database locale che identifica un database remoto, un percorso di comunicazione al database remoto e, optionalmente, il nome di un utente e una password per esso. Una volta definito, il link di database viene utilizzato per eseguire query su tabelle nel database remoto. *Vedere* il Capitolo 21.

DATABASE, FILE

Vedere DATI, FILE.

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 init.ora.

DATABASE, OGGETTO

L'oggetto di un database è qualcosa creato e registrato 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, SPECIFICA

La specifica di un database è un codice alfanumerico che identifica un database di ORACLE, utilizzato per indicare il database nelle operazioni di SQL*NET e per definire il collegamento a un database.

DATABASE, SISTEMA

Un sistema di database è una combinazione di un'istanza e un database. Se l'istanza è avviata e collegata a un database aperto, il sistema è disponibile per l'accesso da parte degli utenti.

DATABASE ADMINISTRATOR (DBA)

Un DBA è un utente di ORACLE autorizzato a garantire (grant) e revocare (revoke) l'accesso al sistema per gli altri utenti, modificare le opzioni di ORACLE che hanno influenza su tutti gli utenti ed eseguire altre funzioni amministrative.

DATABASE APERTO

Un database aperto è un database accessibile dagli utenti.

DATABASE CHIUSO

Un database chiuso è un database associato ad 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 raccolta di database che possono essere gestiti separatamente pur condividendo informazioni.

DATABASE FRAMMENTATO

Un database si dice frammentato quando, essendo stato utilizzato per molto tempo, i dati contenuti nelle tabelle risultano divisi da blocchi vuoti (a seguito di cancellazioni) di dimensioni diverse. La frammentazione rende meno efficiente la gestione dello spazio; questo problema può essere risolto esportando e reimportando alcuni (o tutti) i dati del database.

DATABASE LOCALE

Un database locale è in genere un database che si trova su un computer host. *Vedere per confronto DATABASE REMOTO.*

DATABASE REMOTO

Un database remoto è un database che risiede su un computer remoto. Vi si accede attraverso una connessione di rete.

DATATYPE

Vedere DATI, TIPI

DATAZIONE GIULIANA

La datazione giuliana è un metodo di conversione per cui qualsiasi data può essere espressa utilizzando un unico numero intero. Le date giuliane possono essere ottenute utilizzando la maschera 'J' nelle funzioni i cui argomenti sono date. *Vedere DATE, FORMATI.*

DATE

DATE è uno tipo di dati standard di ORACLE che registra dati di tipo data e ora. Il formato standard per le date è 01-APR-98. Una colonna DATE può contenere una data e un'ora tra il primo gennaio del 4712 A.C. e il trentuno dicembre del 4712 D.C.

DATE, FORMATI

TIPO Parametri di funzioni SQL

PRODOTTI Tutti

VEDERE ANCHE DATE, FUNZIONI; Capitolo 8

DESCRIZIONE I seguenti formati di date vengono utilizzati sia con TO_CHAR che con TO_DATE.

| FORMATO | SIGNIFICATO |
|---------|---|
| MM | Numero di mesi: 12. |
| RM | Numerazione romana dei mesi: XII. |
| MON | Abbreviazione di tre lettere del mese: AUG. |
| Mon | Analogo a MON, ma con la lettera iniziale maiuscola: Aug. |
| mon | Analogo a MON, ma tutto minuscolo: aug. |
| MONTH | Mese : AUGUST. |
| Month | Mese con la lettera iniziale maiuscola: August. |
| month | Mese tutto minuscolo: august. |
| DDD | Numero del giorno nell'anno, da Jan 1: 354. |
| DD | Numero del giorno nel mese: 23. |
| D | Numero del giorno nella settimana: 6. |
| DY | Abbreviazione del giorno a tre lettere: FRI. |
| Dy | Analogo a DY, ma con la lettera iniziale maiuscola: Fri. |
| dy | Analogo a DY, ma tutto minuscolo: fri. |
| DAY | Giorno scritto per esteso: FRIDAY. |
| Day | Giorno con la lettera iniziale maiuscola: Friday. |
| day | Giorno scritto con lettere minuscole: friday. |
| YYYY | Anno scritto per esteso a quattro cifre: 1946. |
| SYYYY | Anno con segno, se B.C. 01000 B.C. = -1000. |
| IYYY | Standard ISO per l'anno a quattro cifre. |

| FORMATO | SIGNIFICATO |
|----------------|--|
| YYY | Ultime tre cifre dell'anno: 946. |
| IYY | Ultime tre cifre dell'anno in formato ISO. |
| YY | Ultime due cifre dell'anno: 46. |
| IY | Ultime due cifre dell'anno ISO. |
| Y | Ultima cifra dell'anno: 6. |
| I | Ultima cifra dell'anno ISO. |
| RR | Ultime due cifre dell'anno, può essere in un altro secolo. |
| YEAR | Anno scritto per esteso:NINETEEN-FORTY-SIX. |
| Year | Scritto per esteso con lettere iniziali maiuscole: Nineteen-Forty-Six. |
| year | Anno il lettere minuscole: nineteen-forty-six. |
| Q | Numero di trimestri: 3. |
| WW | Numero di settimane nell'anno: 46. |
| W | Numero di settimane in un mese: 3. |
| IW | Settimane dell'anno dallo standard ISO. |
| J | "Giuliano"— giorni dal 31 Dicembre, 4713 B.C.: 02422220. |
| 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. |
| /:-. | 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. |
| a.m. | Identico a A.M. ma minuscolo. |
| P.M. | Stesso effetto di A.M. |
| p.m. | Stesso effetto di A.M. |
| AM | Stesso effetto di A.M. ma senza punti. |
| am | Stesso effetto di A.M. ma senza punti. |
| PM | Stesso effetto di A.M. ma senza punti. |
| pm | Stesso effetto di A.M. ma senza punti. |
| CC o SCC | Secolo; S fa precedere BC da "-". |
| B.C. | Visualizza B.C. o A.D. a seconda della data. |
| A.D. | Identico a B.C. |

| FORMATO | SIGNIFICATO |
|---------|---------------------------------|
| b.c. | Identico a B.C. ma minuscolo. |
| a.d. | Identico a B.C. |
| BC o AD | Identico a B.C. ma senza punti. |
| bc o ad | Identico a B.C. ma senza punti. |

I formati di data seguenti funzionano solo con TO_CHAR. Non funzionano con TO_DATE.

| FORMATO | SIGNIFICATO |
|-----------|--|
| "stringa" | stringa è incorporata nella visualizzazione per TO_CHAR. |
| fm | Prefisso per il mese e il giorno: fmMONTH o fmday. 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. |
| 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 qualunque 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 or 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

TIPO Comando SQL

PRODOTTI Tutti, PL/SQL

VEDERE ANCHE DATE, FORMATI; Capitolo 8

DESCRIZIONE Di seguito viene riportato un elenco in ordine alfabetico di tutte le funzioni per date attualmente disponibili in SQL di ORACLE. Ciascuna di esse è elencata in questa guida di riferimento sotto il proprio nome, con il formato esatto e le modalità di utilizzo.

| FUNZIONE | DEFINIZIONE |
|--|---|
| ADD_MONTHS(<i>data,conta</i>) | Aggiunge a <i>data</i> un numero di mesi pari a <i>conta</i> . |
| GREATEST(<i>data1,data2,data3,...</i>) | Restituisce la data più recente da un elenco di date. |
| LAST_DAY(<i>data</i>) | Fornisce la data dell'ultimo giorno del mese in cui <i>data</i> è compreso. |

| FUNZIONE | DEFINIZIONE |
|--------------------------------------|--|
| MONTHS_BETWEEN(<i>data2,data1</i>) | Fornisce <i>data2-data1</i> in mesi (possono essere frazioni di mese). |
| NEXT_DAY(<i>date,'giorno'</i>) | Fornisce la data del ' <i>giorno</i> ' successivo dopo <i>data</i> . |
| NEW_TIME('data','questo','altro') | <i>data</i> è la data (e ora) nel fuso orario corrente. <i>questo</i> è l'abbreviazione a tre lettere del fuso orario corrente. <i>altro</i> quella del fuso orario per il quale si desidera sapere 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 |
| ROUND(<i>data,'formato'</i>) | Arrotonda una data a 12 A.M. (mezzanotte, l'inizio del giorno) se l'ora della data è prima di mezzogiorno, oppure la arrotonda (round) al giorno successivo o in accordo con il formato. Vedere ROUND, più avanti, per i dettagli sul 'formato'. |
| TO_CHAR(<i>data,'formato'</i>) | Riformatta la data secondo <i>formato</i> (vedere DATE, FORMATI). |
| TO_DATE(<i>stringa,'formato'</i>) | Converte una stringa con un certo formato in una data di ORACLE. Accetta anche un numero al posto della stringa con certi limiti. ' <i>formato</i> ' è ristretto. |
| TRUNC(<i>data,'formato'</i>) | Imposta una data a 12 A.M. (mezzanotte, l'inizio del giorno) o in accordo con il formato. Si faccia riferimento a TRUNC per i dettagli riguardanti il formato. |

DATI, BLOCCHI

Quando un utente esegue un'istruzione SQL, i dati a cui l'istruzione fa riferimento vengono bloccati in una certa modalità. L'utente può inoltre bloccare i dati in modo esplicito con un'istruzione LOCK.

DATI, FILE

Un file di dati è semplicemente un file utilizzato per memorizzare i dati in un database. Un database è composto da uno o più tablespace, che a loro volta sono composti da uno o più file di dati.

DATI, INDIPENDENZA

L'indipendenza dai dati è la proprietà di tabelle ben definite che consentono la modifica della struttura fisica e logica di una tabella senza che vengano influenzate le applicazioni che vi accedono.

DATI, TIPI

TIPO Definizioni di dati

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; CONVERSIONE, FUNZIONI; DATE, FUNZIONI; FUNZIONI DI ELENCO; NUMERI, FUNZIONI; ALTRE FUNZIONI

DESCRIZIONE Quando viene creata una tabella è necessario definire le colonne all'interno di essa, ciascuna delle quali appartiene a un certo tipo di dato. 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 dei seguenti tipi.

| TIPO DI DATO | DEFINIZIONE |
|--|--|
| CHAR(<i>dimensione</i>) | Dati di caratteri di lunghezza fissa, pari a <i>dimensione</i> . La dimensione massima è 2000. Quella di default è di 1 byte. Viene effettuato un riempimento a destra con spazi vuoti fino alla lunghezza prefissata. |
| DATE | Intervallo di dati valide da January 1, 4712 B.C. a December 31, 4712 D.C. |
| DECIMAL | Identico a NUMBER. Non accetta come argomento dimensioni o cifre decimali. |
| FLOAT | Identico a NUMBER. |
| INTEGER | Identico a NUMBER. Non accetta come argomento dimensioni o cifre decimali. |
| INTEGER(<i>dimensione</i>) | Intero composto da cifre di <i>dimensione</i> specificata. |
| LONG | Dati di caratteri a dimensione variabile fino a 2Gb di lunghezza. Per ogni tabella, è possibile definire una sola colonna LONG. Le colonne LONG possono non essere utilizzate in sottoquery, funzioni, espressioni, clausole where o indici. Una tabella contenente una colonna LONG può non essere associata a cluster. |
| LONG RAW | Dati binari; oppure identico a LONG. |
| LONG VARCHAR | Identico a LONG. |
| MLSLABEL | Rappresentazione a 4 byte di un'etichetta di sicurezza del sistema operativo. |
| NUMBER | Per colonne NUMBER con spazio per 40 cifre, più lo spazio per una virgola decimale e il segno. I numeri possono essere espressi in due modi: primo, i numeri da 0 a 9, i segni + e - e una virgola decimale; secondo, in notazione scientifica, quale 1.85E3 per 1850. I valori validi sono 0 e numeri positivi e negativi con ampiezza da 1.0E-130 a 9.99.. E125. |
| NUMBER(<i>dimensione</i>) | Per colonne NUMBER di dimensioni specifiche. |
| NUMBER(<i>dimensione</i> , <i>d</i>) | Per colonne NUMBER di <i>dimensione</i> specifica, con una cifra <i>d</i> dopo la virgola decimale. Per esempio NUMBER(5,2) non può contenere un valore maggiore di 999.99. |
| NUMBER(*) | Identico a NUMBER. |
| SMALLINT | Identico a NUMBER. |
| RAW(<i>dimensione</i>) | Dati binari, lunghi un numero di byte pari a <i>dimensione</i> . La dimensione massima è 255 byte. |
| RAW MLSLABEL | Formato binario per etichette di sicurezza del sistema operativo. |

| TIPO DI DATO | DEFINIZIONE |
|-------------------------------|--|
| ROWID | Valore che identifica in modo univoco una riga in un database di ORACLE. Viene restituito dalla pseudocolonna ROWID. |
| VARCHAR2(<i>dimensione</i>) | Stringa di caratteri di lunghezza variabile, con una <i>dimensione</i> massima (fino a 4000). |
| VARCHAR(<i>dimensione</i>) | Identico a VARCHAR2. Utilizzare VARCHAR2, poiché l'utilizzo di VARCHAR può variare nelle versioni future di ORACLE. |
| BLOB | Oggetto binario di grandi dimensioni, fino a 4 Gb di lunghezza. |
| CLOB | Oggetto di caratteri, di grandi dimensioni, fino a 4 Gb di lunghezza. |
| NCLOB | Identico a CLOB, ma per insiemi di caratteri a più byte. |
| BFILE | Puntatore a un file binario del sistema operativo. |

DATO

Singolare della parola dati; un dato rappresenta una singola unità di dati.

DBA

Vedere DATABASE, AMMINISTRATORE (DBA).

DBA, PRIVILEGI

I privilegi per il database vengono assegnati attraverso il comando grant dba e dovrebbero essere limitati a pochissimi utenti.

DBWR, PROCESSO

Uno dei processi in background utilizzato da ORACLE. Il processo Database Writer scrive nuovi dati nel database.

DCL

Vedere DATA CONTROL LANGUAGE (DCL), ISTRUZIONI.

DDL

Vedere DATA DEFINITION LANGUAGE (DDL), ISTRUZIONI.

DDL, BLOCCO

Quando un utente esegue un comando DDL, ORACLE blocca gli oggetti a cui fa riferimento il comando tramite blocchi di DDL. Un blocco DDL blocca gli oggetti nel dizionario dei dati (si confrontino con i blocchi di analisi, l'altro tipo di blocchi del 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 avviene perché ciascun processo trattiene una risorsa che l'altro richiede (quale 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 22.

DECLARE CURSOR (forma 1, interno SQL)

TIPO Comando SQL di dichiarazione per il precompilatore

PRODOTTO Precompilatori

VEDERE ANCHE CLOSE, DECLARE DATABASE, DECLARE STATEMENT, FETCH, OPEN, PREPARE, SELECT, SELECT (racchiuso in SQL)

SINTASSI

```
EXEC SQL [AT {database | :variabile_host}]
    DECLARE cursore CURSOR
        FOR {SELECT comando | istruzioni}
```

DESCRIZIONE L'istruzione DECLARE CURSOR deve comparire prima di qualsiasi comando SQL che faccia riferimento a questo cursore e deve essere compilata nello stesso sorgente delle procedure che fanno riferimento a essa. Il suo nome deve essere unico nel sorgente.

database è il nome di un database già dichiarato tramite un'istruzione DECLARE DATABASE e utilizzato in un comando CONNECT di SQL; *variabile_host* è una variabile che ha come valore tale nome. *cursore* è il nome assegnato a questo cursore. Il comando SELECT è una query con nessuna clausola INTO. In alternativa, può essere utilizzata un'istruzione SQL o un blocco PL/SQL già dichiarato in un'istruzione DECLARE STATEMENT di SQL.

Quando, nell'istruzione SELECT, viene inclusa la clausola FOR UPDATE OF, un'operazione di update 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 di FETCH.

ESEMPIO

```
exec sql at TALBOT declare FRED cursor
    for select Data, Persona, Oggetto, Importo
        from LEDGER
        where Item = :Item
            and Data = :Data
            for update of Importo
```

DECLARE CURSOR (forma 2, PL/SQL)

TIPO Istruzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE CLOSE, FETCH, OPEN, SELECT..INTO, Capitolo 22

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 in via di elaborazione. DECLARE CURSOR definisce un nome per un cursore e dichiara che esso è (IS) il risultato di una certa *istruzione_select*. L'*istruzione_select* è necessaria e può 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 noti 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 voglia modificare la riga corrente all'interno del cursore utilizzando il comando update o il comando delete con la clausola CURRENT OF. Vedere il Capitolo 22 per esempi di processo del cursore all'interno di blocchi PL/SQL.

DECLARE DATABASE (interno SQL)

TIPO Comando SQL di dichiarazione per il precompilatore

PRODOTTO Precompilatori

VEDERE ANCHE COMMIT RELEASE (interno SQL), CONNECT (Interno SQL), *Programmer's Guide to the ORACLE Precompilers*

SINTASSI EXEC SQL DECLARE *database* DATABASE

DESCRIZIONE DECLARE DATABASE dichiara il nome di un database remoto per utilizzarlo in successive clausole AT all'interno di istruzioni SQL, incluse COMMIT, DECLARE CURSOR, DELETE, INSERT, ROLLBACK, SELECT e UPDATE.

DECLARE STATEMENT (interno SQL)

TIPO Comando SQL di dichiarazione per il precompilatore

PRODOTTO Precompilatori

VEDERE ANCHE CLOSE, FETCH, OPEN, PREPARE

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, che deve essere identica all'istruzione presente qui. *database* è il nome di un database dichiarato precedentemente tramite DECLARE DATABASE; *variabile_host* può contenere come valore tale nome. Questo comando è necessario solo se DECLARE CURSOR è specificato prima di PREPARE. Quando si utilizza, dovrebbe essere collocato prima di DECLARE, DESCRIBE, OPEN o PREPARE e deve essere compilato nello stesso sorgente delle procedure che vi fanno riferimento.

DECLARE TABLE

TIPO Comando del precompilatore

PRODOTTO Precompilatori

VEDERE ANCHE CREATE TABLE

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'utilizzo di NULL e NOT NULL.

Si utilizza DECLARE TABLE per comunicare ai precompilatori di ignorare le definizioni delle tabelle reali del database di ORACLE quando si avvia un'esecuzione con SQLCHECK=FULL. I precompilatori considerano come rilevante per il programma la descrizione della tabella fornita qui e ignorano le definizioni delle tabelle nel database. È di utilità quando le definizioni delle tabelle sono destinate a cambiare o quando una tabella non è stata ancora creata. Se SQLCHECK non è uguale a FULL (il che significa che le tabelle e le colonne non vengono comunque controllate con il database), questo comando viene ignorato dal precompilatore e diventa semplicemente una documentazione.

ESEMPIO

```
EXEC SQL DECLARE COMFORT TABLE (
    Citta          VARCHAR2(13) NOT NULL,
    DataEsempio   DATE NOT NULL,
    Mezzogiorno   NUMBER(3,1),
    Mezzanotte    NUMBER(3,1),
    Precipitazione NUMBER
);
```

DECODE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTRE FUNZIONI, TRANSLATE, Capitolo 16

SINTASSI DECODE(*valore,if1,then1[,if2,then2,]... ,else*)

DESCRIZIONE *valore* rappresenta qualsiasi colonna in una tabella, indipendentemente 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 corrisponde a nessuno degli *if*, il risultato di DECODE è *else* e il valore di *else* può anche essere una colonna o il risultato di una funzione o di un calcolo. Il Capitolo 16 è dedicato interamente a DECODE.

ESEMPIO

```
select DISTINCT Citta,
    DECODE(Citta, 'SAN FRANCISCO', 'Citta DELLA BAIA', Citta)
    from COMFORT;
    Citta      DECODE(Citta, 'SA
    -----
    PARIGI      PARIGI
    SAN FRANCISCO Citta DELLA BAIA
```

DEFAULT

Default è una clausola o il valore di un'opzione che viene utilizzato se non risulta specificata alcuna alternativa. Si può specificare il valore di default per una colonna di una tabella tramite la limitazione sulla colonna DEFAULT.

DEFAULT, VALORE

Il valore di default è un valore che viene utilizzato se non ne viene specificato o inserito esplicitamente un altro.

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

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE APPEND, CHANGE, EDIT, INPUT, LIST, Capitolo 5

SINTASSI DEL

DESCRIZIONE DEL elimina la riga corrente del buffer attivo in quel momento. Con un unico comando DEL si possono cancellare più righe.

ESEMPIO Elenco dei contenuti del buffer corrente:

list

```
1 select Persona, Importo
2   from REGISTRO
3 where Importo > 10000
4*   and Tasso = 3;
```

L'asterisco mostra che la riga corrente è la 4. Per cancellare la seconda riga è necessario immettere il comando:

list 2

```
2*   from REGISTRO
```

che fa in modo che la riga corrente diventi la numero 2. Successivamente si immette il comando:

del

per eliminare la riga. La vecchia riga numero 3 ora risulta essere la riga numero 2, la riga 4 è diventata la 3 e così via.

Per eliminare una serie di righe 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)

TIPO Comando SQL

PRODOTTI PL/SQL

VEDERE ANCHE DECLARE CURSOR, UPDATE, Capitolo 22

DESCRIZIONE In PL/SQL, DELETE segue le regole del comando SQL normale, con le funzionalità aggiuntive descritte di seguito.

- Si può utilizzare nella clausola where una funzione e/o variabile PL/SQL, proprio come una qualunque sequenza di lettere.
- DELETE . . WHERE CURRENT OF cursor può essere utilizzato assieme 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, 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 risulta uguale a 0, significa che nessuna riga è stata eliminata (inoltre, se nessuna riga è stata eliminata, SQL%FOUND risulta essere uguale a FALSE)

ESEMPIO Per cancellare tutti gli impiegati di 65 anni o più dalla tabella IMPIEGATI (perché, per esempio, sono andati in pensione), si utilizzi:

```

DECLARE
    cursor IMPIEGATI is
        select Eta from LAVORATORE
        for update of Eta;

    LAVORATORE_RECORD    LAVORATORE%ROWTYPE;

BEGIN
    open IMPIEGATI;
    loop
        fetch LAVORATORE into LAVORATORE_RECORD;
        exit when IMPIEGATI%NOTFOUND;
        if LAVORATORE_RECORD.Eta >= 65
            then delete LAVORATORE where current of IMPIEGATI;
    end loop;
    close IMPIEGATI;

END;

```

DELETE (forma 2, Comando SQL)

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE DROP TABLE, FROM, INSERT, SELECT, UPDATE, WHERE, Capitolo 14

SINTASSI DELETE FROM [utente.]tabella[@link] [alias] [WHERE condizione];

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, deve essere definito un link di database tramite @link. Se viene specificato link, ma non utente, la query cerca una tabella di proprietà dell'utente sul database remoto.

ESEMPIO Nell'esempio seguente vengono eliminate tutte le righe della Citta di Parigi dalla tabella COMFORT:

```

delete from COMFORT
where Citta = 'PARIGI';

```

DELETE (forma 3, Interno SQL)

TIPO Comando del precompilatore

PRODOTTO Precompilatori

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 per il precompilatore).

Deref

L'operatore **DEREF** restituisce il valore di un oggetto di riferimento. Vedere Capitolo 31.

DESCRIBE (forma 1, Comando SQL*PLUS)

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE CREATE TABLE

SINTASSI DESC[RIBE] [utente.]tabella

DESCRIZIONE DESCRIBE visualizza la definizione della tabella specificata. Se viene omesso *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 dato e l'ampiezza o precisione.

ESEMPIO describe COMFORT

| Name | Null? | Type |
|----------------|----------|--------------|
| Citta | NOT NULL | VARCHAR2(13) |
| ESEMDATA | NOT NULL | DATE |
| MEZZOGIORNO | | NUMBER(3,1) |
| MEZZANOTTE | | NUMBER(3,1) |
| PRECIPITAZIONE | | NUMBER |

DESCRIBE (forma 2, Interno SQL)

TIPO Comando per il precompilatore

PRODOTTO Precompilatori

VEDERE ANCHE **PREPARARE**

SINTASSI EXEC SQL DESCRIBE [BIND VARIABLES FOR
| SELECT LIST FOR]
| { *nome_istruzione*
| *nome_blocco* } INTO *descrittore*

DESCRIZIONE Il comando DESCRIBE inizializza un descrittore in modo che contenga le descrizioni delle variabili dell'host per un'istruzione SQL dinamica o un blocco PL/SQL. Il comando DESCRIBE segue il comando PREPARE.

DESCRIZIONE, FASE

La fase di descrizione è una fase dell'esecuzione di una query SQL, in cui il programma raccoglie informazioni riguardo ai risultati della query da eseguire.

DEVICE

Device è il termine in inglese utilizzato per indicare un terminale video (CRT), quale un IBM 3270, DEC VT100 o VT200.

DICHIARAZIONI, ISTRUZIONI

Un'istruzione SQL di dichiarazione è un'istruzione che non genera una chiamata al database e quindi non è un'istruzione SQL eseguibile. Esempi al riguardo sono BEGIN DECLARE SECTION o DECLARE CURSOR. Le istruzioni di dichiarazione vengono utilizzate principalmente nei programmi PL/SQL e precompilati. Si faccia il confronto con le istruzioni SQL eseguibili.

DIFFERITO, SEGMENTO DI ROLLBACK

Un segmento di rollback differito è un segmento contenente voci che non possono essere applicate al tablespace, perché quest'ultimo risulta scollegato. Nel momento in cui il tablespace ritorna a essere disponibile, tutte le voci vengono applicate.

DIPENDENTI, PARAMETRI

Sono parametri di init.ora che non devono essere modificati dagli utenti o dai DBA perché ORACLE calcola automaticamente i loro valori in base ai valori di uno o più parametri di init.ora.

DIRECTORY

Un nome logico che punta a una directory fisica, utilizzata dai LOB di BFILE. Vedere CREATE DIRECTORY.

DISABLE

TIPO Clausola di comandi SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER TABLE, CREATE TABLE, ENABLE, VINCOLO DI INTEGRITÀ

SINTASSI DISABLE {{UNIQUE(*colonna*[,*colonna*]...) |
 PRIMARY KEY |
 CONSTRAINT *limitazione*} [CASCADE]} |
 ALL TRIGGERS}

DESCRIZIONE La clausola DISABLE disabilita un vincolo di integrità o un trigger. UNIQUE disabilita un vincolo unico particolare. PRIMARY KEY disabilita il singolo vincolo della chiave primaria. CONSTRAINT disabilita il vincolo di integrità specificata. CASCADE disabilita qualsiasi vincolo di integrità che dipenda dai vincoli specificati. Quindi, se esiste un vincolo di integrità referenziale definito in un'altra tabella che si riferisce a una chiave primaria, o unica, in questa tabella, è necessario specificare CASCADE per disabilitare il vincolo della chiave unica o primaria.

In un'istruzione ALTER TABLE, è possibile specificare ALL TRIGGERS per disabilitare tutti i trigger associati alla tabella. Non è possibile utilizzare questa clausola in CREATE TABLE.

È inoltre possibile disabilitare un vincolo in CREATE TABLE specificando la parola DISABLE nella clausola CONSTRAINT. Vedere VINCOLO DI INTEGRITÀ.

DISCONNECT

TIPO Comando SQL*PLUS

PRODOTTO SQL*PLUS

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 cambiare i contenuti dei buffer, utilizzare il proprio editor, eseguire lo spool 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 quando esegue EXIT o QUIT, l'utente risulta ancora collegato, viene automaticamente scollegato e fatto uscire da ORACLE.

DISTACCATO, PROCESSO

Vedere BACKGROUND, PROCESSO.

DIZIONARIO di DATI

Il dizionario di dati rappresenta un elenco completo dell'insieme delle tabelle e delle viste di proprietà degli utenti SYS e SYSTEM del DBA, che viene attivato quando viene inizialmente installato ORACLE ed è una fonte centrale di informazioni per gli RDBMS stessi di ORACLE e per tutti gli utenti. Le tabelle vengono automaticamente gestite da ORACLE e contengono un insieme di viste e tabelle comprendenti informazioni relative agli oggetti, agli utenti, ai privilegi, agli eventi e all'utilizzo del database. Vedere il Capitolo 32.

DIZIONARIO, BLOCCHI

Un blocco relativo a un dizionario è un blocco condiviso, di proprietà di utenti che stanno analizzando istruzioni DML, oppure un blocco esclusivo di proprietà degli utenti che stanno eseguendo comandi DDL, per evitare che una tabella venga alterata mentre si stanno interrogando i dizionari. Possono esistere più blocchi di questo tipo contemporaneamente.

DIZIONARIO, BLOCCHI DELLA CACHE

I blocchi della cache del dizionario sono uno dei tre tipi di blocchi interni sulle voci presenti nelle cache del dizionario.

DIZIONARIO, BLOCCHI OPERATIVI

Un blocco di tipo operativo su un dizionario è un blocco esclusivo attivo durante la fase di aggiornamento del dizionario. Ne esiste solo uno ed è sempre esclusivo.

DIZIONARIO, CACHE

La cache del dizionario è una delle diverse cache di informazioni riguardanti il dizionario di dati, contenute nella SGA. Le informazioni del dizionario presenti nella cache migliorano le prestazioni, se le informazioni del dizionario vengono utilizzate spesso.

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**.

DML

Vedere le istruzioni del linguaggio di manipolazione dei dati DML.

DML LOCK

DML lock è un sinonimo di “blocco sui dati”.

DOCUMENT

TIPO Comando di SQL*PLUS

PRODOTTO SQL*PLUS

VEDERE ANCHE #, /* */, REMARK

SINTASSI DOC[UMENT]

DESCRIZIONE DOCUMENT informa SQL*PLUS che sta cominciando un blocco di documentazione. Il simbolo di # posto su una riga a sé termina il blocco. Per funzionare a dovere DOCUMENT deve essere stato attivato. In caso contrario SQL*PLUS tenta di eseguire le righe comprese tra DOCUMENT e il simbolo #. Questo costrutto viene utilizzato per eseguire o meno una sequenza di istruzioni sulla base di risultati precedenti (un file di configurazione in cui si sia impartito SET DOCUMENT OFF/ON, o quando siano state utilizzate le tecniche di spooling e avvio descritte nel Capitolo 20).

Se si è scelto di reindirizzare l'output di una sessione interattiva verso un file, digitando l'istruzione DOCUMENT il prompt di SQL cambia da SQL> a DOC>. Tutto quello che viene digitato prima dell'inserimento del carattere # finisce nel file senza che SQL*PLUS cerchi di eseguirlo.

SQL*PLUS visualizza le righe contenute nelle sezioni DOCUMENT a meno che TERMOUT sia impostato su OFF.

L'editor della riga di comando di SQL*PLUS non è in grado di bufferizzare direttamente un simbolo # quando è la prima lettera di una riga. Per inserirlo si digiti un altro carattere, si inserisca il cancelletto e quindi si cancelli il carattere inutile.

DOCUMENT è un comando obsoleto; al suo posto si consiglia di utilizzare l'istruzione REMARK.

ESEMPIO Se DOCUMENT fosse stato impostato su OFF:

```
DOCUMENT
column password print
```

```
REM cambia la visualizzazione della password
#
```

DROP CLUSTER

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE CLUSTER, DROP TABLE, Capitolo 19

SINTASSI `DROP CLUSTER [utente.]cluster
[INCLUDING TABLES [CASCADE CONSTRAINTS]]`

DESCRIZIONE `DROP CLUSTER` elimina un cluster dal database. Se il cluster non fa parte del proprio schema è necessario che l'utente abbia il privilegio `DROP ANY CLUSTER`. `DROP CLUSTER` salva eventuali modifiche pendenti nel database. Non è possibile eliminare un cluster che contiene delle tabelle: è necessario eliminarle per prime. L'opzione `INCLUDING TABLES` consente di cancellare automaticamente tutte le tabelle contenute all'interno dei cluster. L'opzione `CASCADE CONSTRAINTS` elimina tutti i vincoli riguardanti l'integrità dei riferimenti esterni contenuti come chiavi all'interno delle tabelle suddivise in cluster.

Non è possibile rimuovere da un cluster le tabelle in modo individuale. Per ottenere lo stesso effetto, è necessario copiare la tabella con un nuovo nome (si utilizzi `CREATE TABLE` assieme ad `AS SELECT`), eliminare quello vecchio (rimuovendolo così dal cluster), assegnare alla copia lo stesso nome di quella appena eliminata (mediante l'istruzione `RENAME`) impartire i privilegi opportuni tramite `GRANT` e creare gli indici necessari.

DROP DATABASE LINK

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE DATABASE LINK, Capitolo 21

SINTASSI `DROP [PUBLIC] DATABASE LINK link`

DESCRIZIONE `DROP DATABASE LINK` elimina un link da un database di proprietà dell'utente. Nel caso si tratti di un link pubblico, è necessario utilizzare l'opzione `PUBLIC` (occorre essere un DBA per poterla impartire). `PUBLIC` non può essere utilizzata per eliminare un collegamento privato. *link* è il nome del link da eliminare. Per poter eliminare un link pubblico da un database è necessario possedere il privilegio di sistema `DROP PUBLIC DATABASE LINK`. All'interno del proprio spazio è permesso eliminare link privati.

ESEMPIO La riga seguente elimina il link di nome `SEDE_DI_ANA` dal database corrente:

```
drop database link SEDE_DI_ANA;
```

DROP DIRECTORY

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE DIRECTORY, Capitolo 27

SINTASSI `DROP DIRECTORY nome_directory`

DESCRIZIONE `DROP DIRECTORY` elimina una directory (utilizzata dai tipi `BFILE`). Per poter eliminare una directory è indispensabile avere il privilegio `DROP ANY DIRECTORY`.

ESEMPIO La riga seguente elimina la directory di nome PROPOSTE:

```
drop directory PROPOSTE;
```

DROP FUNCTION

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER FUNCTION, CREATE FUNCTION, Capitolo 24

SINTASSI DROP FUNCTION [*utente*.]*funzione*

DESCRIZIONE DROP FUNCTION elimina la funzione specificata. Questa istruzione salva eventuali modifiche pendenti nel database. ORACLE invalida qualsiasi oggetto che dipenda o che richiami la funzione in questione. Per eliminare una funzione non in proprio possesso è necessario avere il privilegio di sistema DROP ANY PROCEDURE.

DROP INDEX

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER INDEX, CREATE INDEX, CREATE TABLE, Capitolo 19

SINTASSI DROP INDEX [*utente*.]*indice*

DESCRIZIONE DROP INDEX elimina l'indice specificato. È necessario essere i proprietari dell'indice specificato o avere l'attribuito di sistema DROP ANY INDEX.

DROP LIBRARY

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE LIBRARY, Capitolo 24

SINTASSI DROP LIBRARY [*utente*,]*libreria*

DESCRIZIONE DROP LIBRARY elimina la libreria specificata. È necessario possedere il privilegio di sistema DROP LIBRARY.

DROP PACKAGE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER PACKAGE, CREATE PACKAGE, Capitolo 24

SINTASSI DROP PACKAGE [BODY] [*utente*.]*package*

DESCRIZIONE DROP PACKAGE elimina il package specificato. Se si utilizza il parametro opzionale BODY si elimina il solo contenuto senza intaccare la specifica del package. Questa istruzione salva eventuali cambiamenti pendenti nel database. ORACLE invalida tutti gli oggetti che dipendano dal package (se si è scelto di cancellare anche le specifiche). Per utilizzare questo comando è necessario essere proprietari del package o possedere il privilegio di sistema DROP ANY PROCEDURE.

DROP PROCEDURE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER PROCEDURE, CREATE PROCEDURE, Capitolo 24

SINTASSI DROP PROCEDURE [*utente.*]*procedura*

DESCRIZIONE DROP PROCEDURE elimina la procedura specificata. Questa istruzione salva eventuali modifiche pendenti nel database. ORACLE invalida tutti gli oggetti che dipendono o che richiamano la procedura in oggetto. Per impartire questa istruzione è necessario essere di proprietari della procedura o possedere l'attributo di sistema DROP ANY PROCEDURE.

DROP PROFILE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER PROFILE, CREATE PROFILE, Capitolo 18

SINTASSI DROP PROFILE [*utente.*]*profilo*

DESCRIZIONE DROP PROFILE elimina il profilo specificato. Questa istruzione salva eventuali modifiche pendenti nel database. È necessario avere il privilegio di sistema DROP PROFILE.

DROP ROLE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER ROLE, CREATE ROLE, Capitolo 18

SINTASSI DROP ROLE [*utente.*]*ruolo*

DESCRIZIONE DROP ROLE elimina il ruolo specificato. Questa istruzione salva eventuali modifiche pendenti nel database. È necessario essere in possesso del privilegio di sistema DROP ANY ROLE.

DROP ROLLBACK SEGMENT

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER ROLLBACK SEGMENT, CREATE ROLLBACK SEGMENT, CREATE TABLESPACE, SHUTDOWN, AVVIO

SINTASSI DROP [PUBLIC] ROLLBACK SEGMENT *segmento*

DESCRIZIONE *segmento* è il nome di un segmento di rollback esistente da cancellare. Il segmento non deve essere attivo all'atto dell'esecuzione dell'istruzione. L'opzione PUBLIC è necessaria quando si devono eliminare seguenti di rollback pubblici.

La modalità di visualizzazione DBA_ROLLBACK_SEGS può discriminare quali seguenti di rollback siano aperti in un dato istante evidenziandoli nella colonna Status. Se il segmento è attualmente utilizzato si può attendere fino a che venga rilasciato o chiudere il database (con SHUTDOWN e IMMEDIATE) per poi aprirlo in modalità esclusiva utilizzando STARTUP. Per eliminare un segmento di rollback è necessario avere il privilegio di DBA.

DROP SEQUENCE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER SEQUENCE, CREATE SEQUENCE, Capitolo 19

SINTASSI DROP SEQUENCE [*utente.*]sequenza

DESCRIZIONE Sequenza è il nome della sequenza da eliminare. Per effettuare questa operazione è necessario possedere il privilegio di sistema DROP ANY SEQUENCE.

DROP SNAPSHOT

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER SNAPSHOT, CREATE SNAPSHOT, Capitolo 28

SINTASSI DROP SNAPSHOT [*utente.*]snapshot

DESCRIZIONE L'istruzione DROP SNAPSHOT elimina lo snapshot indicato. Vengono salvate eventuali modifiche ancora pendenti nel database. I privilegi richiesti sono il possesso o il privilegio di sistema DROP ANY SNAPSHOT. Si rimanda al Capitolo 28 per dettagli riguardanti l'implementazione degli snapshot.

DROP SNAPSHOT LOG

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER SNAPSHOT LOG, CREATE SNAPSHOT LOG, Capitolo 28

SINTASSI DROP SNAPSHOT LOG ON [*utente.*]tabella

DESCRIZIONE DROP SNAPSHOT LOG elimina la tabella di log indicata. Questa operazione effettua una modifica pendente al database. La cancellazione è permessa solo al possessore della tabella e agli utenti in possesso del privilegio di sistema DROP ANY SNAPSHOT. Dopo la cancellazione del log, a ciascun snapshot della tabella principale verrà assegnato l'attributo di aggiornamento COMPLETE togliendo eventualmente gli attributi FAST.

DROP SYNONYM

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE SYNONYM, Capitolo 21

SINTASSI DROP [PUBLIC] SYNONYM [*utente.*]sinonimo

DESCRIZIONE DROP SYNONYM elimina il sinonimo specificato ed effettua una modifica pendente al database. Per eliminare un sinonimo di tipo pubblico è necessario possedere il privilegio di sistema DROP ANY PUBLIC SYNONYM mentre per quelli di tipo privato è sufficiente essere il proprietario o avere il privilegio di sistema DROP ANY SYNONYM.

DROP TABLE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER TABLE, CREATE INDEX, CREATE TABLE, DROP CLUSTER

SINTASSI DROP TABLE [*utente.*]tabella [CASCADE CONSTRAINTS]

DESCRIZIONE DROP TABLE elimina la tabella e salva eventuali modifiche pendenti nel database. Per eliminare una tabella è necessario esserne proprietario o possedere il privilegio di sistema DROP ANY TABLE. Quando viene eliminata una tabella di fatto vengono soppressi anche tutti gli indici e i contenuti ad essa associati. Gli oggetti contenenti tabelle in corso di cancellazione vengono marcati come non validi e cessano di funzionare.

L'opzione CASCADE CONSTRAINTS elimina tutti vincoli riguardanti l'integrità dei riferimenti esterni contenuti nella tabella in corso di cancellazione.

È possibile eliminare un cluster con tutte le tabelle relative utilizzando l'opzione INCLUDING TABLES del comando DROP CLUSTER.

DROP TABLESPACE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER TABLESPACE, CREATE DATABASE, CREATE TABLESPACE, Capitolo 19

SINTASSI `DROP TABLESPACE tablespace [INCLUDING CONTENTS]`

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 vuoti. Prima della cancellazione i tablespace devono essere offline (*vedere* ALTER TABLESPACE) o la cancellazione potrebbe essere evitata dagli altri utenti eventualmente al lavoro sugli stessi dati. Per utilizzare questo comando è necessario disporre del privilegio di sistema DROP TABLESPACE.

DROP TRIGGER

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER TRIGGER, CREATE TRIGGER, Capitolo 23 e Capitolo 25

SINTASSI `DROP TRIGGER [utente.] trigger`

DESCRIZIONE DROP TRIGGER elimina il trigger indicato. Questa istruzione salva eventuali modifiche pendenti nel database. Per eliminare un trigger è necessario disporre del privilegio di sistema DROP ANY TRIGGER.

DROP TYPE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE TYPE, Capitolo 4

SINTASSI `DROP TYPE [utente,] tipo [FORCE]`

DESCRIZIONE DROP TYPE elimina la specifica e il corpo di un tipo di dato astratto. È necessario essere i proprietari del tipo o disporre del privilegio di sistema DROP ANY TYPE. Non è possibile eliminare un tipo di dato se una tabella o qualsiasi altro oggetto vi sta ancora facendo riferimento.

DROP TYPE BODY

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE TYPE, CREATE TYPE BODY, DROP TYPE, Capitolo 25

SINTASSI `DROP TYPE BODY [utente.]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

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER USER, CREATE USER, Capitolo 18

SINTASSI `DROP USER utente [CASCADE]`

DESCRIZIONE `DROP USER` elimina l'utente indicato. Questa istruzione salva eventuali modifiche pendenti nel database. È necessario disporre del privilegio di sistema `DROP USER`. L'opzione `CASCADE` elimina tutti gli oggetti dello 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

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE SYNONYM, CREATE TABLE, CREATE VIEW, Capitolo 17

SINTASSI `DROP VIEW [utente.]vista`

DESCRIZIONE `DROP VIEW` elimina la vista specificata e salva eventuali modifiche pendenti nel database. Solo gli utenti con diritti DBA possono eliminare le viste create da altri utenti. Le viste e i sinonimi che si basano su viste cancellate vengono marcati come inservibili. Le viste da eliminare devono essere di proprietà dell'utente, oppure questo deve possedere il privilegio di sistema `DROP ANY VIEW`.

DUAL

TIPO Tabella di lavoro di ORACLE

PRODOTTI Tutti

VEDERE ANCHE FROM, SELECT, Capitolo 8

DESCRIZIONE `DUAL` è una tabella contenente solo una riga e una colonna. Dato che la maggior parte delle funzioni di ORACLE lavora sia sulle colonne che sui letterali è possibile utilizzare delle pseudocolonne, come nel caso di `SysDate`. In tal caso l'istruzione `select` non verifica quali colonne siano 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 correnti:

```
select User, SysDate from DUAL;
```

DUMP

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE RAWTOHEX

SINTASSI DUMP(*stringa* [,*formato* [,*inizio* [, *lunghezza*]]])

DESCRIZIONE DUMP visualizza il valore di una 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 su cui gira ORACLE; 8 produce la notazione ottale, 10 quella decimale, 16 quella esadecimale mentre 17 rappresenta la modalità carattere (in maniera analoga 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 della prima riga della tabella COMFORT:

```
select Citta, dump(Citta,16,1,8) a 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

TIPO Comando di SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE DEFINE, SET, Capitolo 5

SINTASSI EDIT [*file*[.*est*]]

DESCRIZIONE EDIT richiama un editor di testo esterno e gli passa come argomento il nome del file. Se si omette l'estensione viene aggiunta per default l'estensione 'SQL'. Se si omettono sia il nome che l'estensione del file, l'editor viene richiamato con un nome inventato da SQL*PLUS. Tale file permetterà la modifica del contenuto corrente del buffer.

La variabile locale d'utente _EDITOR determina quale editor di testo debba essere richiamato da EDIT. _EDITOR può essere modificato mediante DEFINE, generalmente all'interno del file LOGIN.SQL che viene eseguito ogni volta che si ricorre a SQL*PLUS.

EDIT provoca un messaggio di errore quando viene richiamato senza parametri e il buffer corrente è vuoto. È possibile utilizzare il comando SET EDITFILE per impostare il nome del file da modificare o da creare per default.

ELABORAZIONE DISTRIBUITA

L'elaborazione distribuita è una computazione effettuata su diverse CPU per ottenere un singolo risultato.

EMBEDDED (SQL*PLUS)

Vedere SET.

ENABLE

TIPO Opzione di un comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER TABLE, CREATE INDEX, CREATE TABLE, DISABLE, VINCOLO DI INTEGRITÀ, Capitolo 32

SINTASSI

```
ENABLE {{UNIQUE(colonna[,colonna]... |
          PRIMARY KEY |
          CONSTRAINT vincolo}|
        [USING INDEX [INITTRANS intero]
         [MAXTRANS intero]
         [TABLESPACE tablespace]
         [STORAGE spazio]
         [PCTFREE intero]]|
        [EXCEPTIONS INTO [utente.]tabella] |
        ALL TRIGGERS}}
```

DESCRIZIONE L'opzione ENABLE abilita un vincolo di integrità o un trigger precedentemente disabilitato. Quando si abilita una regola di tipo UNIQUE o PRIMARY KEY, ORACLE costruisce un indice del colonne della chiave. L'opzione USING INDEX specifica i parametri dell'indice. Se si sceglie di abilitare un vincolo di integrità dei riferimenti è necessario che anche la chiave esterna sia abilitata.

L'opzione EXCEPTIONS INTO specifica il nome di una tabella delle eccezioni esistente da archiviare localmente. Quando ORACLE riconosce una violazione di integrità, l'informazione viene archiviata in questa tabella. Quest'ultima viene creata dallo script UTLEXCPT.SQL (o da quello equivalente sulla piattaforma dell'utente). Vedere il Capitolo 32 per ulteriori dettagli.

L'opzione ALL TRIGGERS può comparire solo all'interno dell'istruzione ALTER TABLE e permette di abilitare tutti i trigger associati alla tabella.

END

TIPO Istruzione di delimitazione di blocchi

PRODOTTI PL/SQL

VEDERE ANCHE BEGIN; BLOCCHI, STRUTTURA; Capitolo 22

SINTASSI END [*blocco*];

DESCRIZIONE END non ha alcuna relazione con le istruzioni END IF ed END LOOP. Vedere IF e LOOP per ulteriori dettagli a riguardo.

END è l'istruzione di chiusura della sezione eseguibile di un blocco PL/SQL. Se nell'istruzione BEGIN è stato assegnato un nome al blocco, questo deve necessariamente seguire la parola END. Tra BEGIN ed END è richiesta la presenza di almeno un'istruzione eseguibile. Vedere il Capitolo 22 per ulteriori dettagli.

ENQUEUE

L'enqueue (accodamento) è un meccanismo di limitazione degli accessi di una particolare risorsa. Gli utenti o i processi che necessitano della disponibilità della risorsa in questione attendono che il sistema fornisca loro l'enqueue della risorsa. La

maggior parte delle risorse contenute nei database consente l'utilizzo di questo meccanismo.

ENTITÀ

Una entità è una persona, una località geografica o un qualsiasi oggetto rappresentabile mediante una tabella nella quale ciascuna riga rappresenti una sua occorrenza.

ENTITÀ-RELAZIONI, MODELLO

Un modello entità-relazioni è un costrutto utilizzato nella realizzazione di modelli di sistemi di database. Questo modello divide tutti gli elementi di un sistema in due categorie: le entità e le relazioni. Un semplice esempio è riportato nella Figura 33.1 del Capitolo 33.

EQUI-JOIN

Un equi-join è un'unione in cui l'operatore di confronto è un simbolo di uguaglianza. Ad esempio:

```
where LAVORATORE.Nome = COMPITOLAVORATORE.Nome
```

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 utilizzando le lettere da A a F. Questo sistema viene utilizzato spesso nella visualizzazione dei dati interni ai sistemi di elaborazione.

ESCAPE (SQL*PLUS)

Vedere SET.

ESPRESSIONE

Un'espressione è un qualsiasi tipo di campo o 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 unico valore, quale ad 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.

ESTENSIONI LIBERE

Le estensioni libere sono blocchi di spazio non ancora allocati per alcuna tabella o segmento. *Estensioni libere* è quindi sinonimo di spazio libero.

ESTENSIONI USATE

Le cosiddette "estensioni usate" sono le zone di memoria che sono già state allocate (o riservate) a segmenti di dati. Vede ESTENSIONI LIBERE.

EXCEPTION

TIPO Istruzione di delimitazione di sezioni

PRODOTTI PL/SQL**VEDERE ANCHE** RAISE, Capitolo 22**SINTASSI** EXCEPTION

```
{WHEN {OTHERS | eccezione [OR eccezione]}...}
THEN istruzione; [istruzione;]...
[ WHEN {OTHER | eccezione [OR eccezione]}...
THEN istruzione; [istruzione;]...]...
```

DESCRIZIONE La sezione di un blocco PL/SQL che inizia con l'istruzione EXCEPTION è quella a cui passa il flusso dell'esecuzione del programma quando viene identificato un flag di eccezione. Questi possono essere definiti dall'utente o far parte delle eccezioni di sistema eseguite automaticamente da PL/SQL. Vedere RAISE per ulteriori dettagli sui flag di eccezione definibili dall'utente. I flag di eccezione del sistema sono variabili BOOLEAN (che possono essere TRUE oppure FALSE). L'opzione WHEN viene impiegata per la loro valutazione. Ad esempio, il flag NOT_LOGGED_ON viene attivato quando si cerca di impartire un comando ad ORACLE senza aver avuto accesso al sistema. Non è necessario dichiarare questo tipo di eccezione o chiamarne 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. Questa sezione, però, non ha modo di capire automaticamente quale tipo di eccezione sia stato generato. Per questo motivo è necessario includere nel codice di gestione delle eccezioni un controllo che valuti tutti i possibili tipi di eccezioni e decida la strategia da adottare. Questo è quanto viene svolto dal costrutto logico WHEN...THEN. È possibile utilizzare WHEN OTHERS per gestire tutte le eccezioni per cui non si sia effettuato un controllo specifico a livello di WHEN...THEN.

ESEMPIO Nell'esempio che segue, viene valutato un flag di eccezione all'interno di una sezione EXCEPTION. Il flag ZERO_DIVIDE viene attivato quando si cerca di dividere per zero.

```
declare
    pi      constant NUMBER(9,7) := 3.1415926;
    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;
        exit when area >100;
    end loop;
exception
    when ZERO_DIVIDE
        then insert into AREE values (0,0);
end;
/

```

Di seguito è riportato un elenco dei principali flag di eccezione attivati dal sistema.

- **CURSOR_ALREADY_OPEN** viene attivato quando un'istruzione OPEN cerca di aprire un cursore che è già aperto. SQLCODE viene impostato a -6511 (il codice di errore è ORA-06511).
- **DUP_VAL_ON_INDEX** viene attivato quando insert o update provocherebbero un doppione nei valori di un indice. SQLCODE viene impostato a -1 (il codice di errore è ORA-00001).
- **INVALID_CURSOR** viene attivato quando si cerca di aprire un cursore che non sia stato precedentemente dichiarato, quando si cerca di chiuderne uno che è già stato chiuso o quando si cerca di effettuare un FETCH da un cursore chiuso, e così via. SQLCODE viene impostato su -1001 (il codice di errore è ORA-01001).
- **INVALID_NUMBER** viene attivato quando si verifica un errore di conversione da una stringa di caratteri a un numero, quando la stringa di caratteri non contiene un numero corretto. SQLCODE viene impostato a -1722 (il codice di errore è ORA-01722).
- **LOGIN_DENIED** viene attivato quando ORACLE rifiuta il nome dell'utente o la sua password all'atto dell'ingresso nel sistema. SQLCODE viene impostato a -1017 (il codice di errore è ORA-01017).
- **NO_DATA_FOUND** viene attivato quando un'istruzione di selezione non trova alcuna riga che soddisfi i requisiti richiesti. (Non è la stessa eccezione generata da FETCH quando non restituisce alcuna riga. NO_DATA_FOUND è un flag specifico dell'istruzione select). SQLCODE viene impostato su +100 (il codice di errore è ORA-01403). I codici di errore sono differenti in quanto +100 è il codice di errore standard ANSI che indica che i dati non sono stati trovati. Si noti che questo codice di errore è positivo; questo in generale significa che l'errore è recuperabile.
- **NOT_LOGGED_ON** viene attivato quando si cerca di effettuare qualsiasi tipo di operazione su un database senza essersi precedentemente collegati. SQLCODE viene impostato a -1012 (il codice di errore è ORA-01012).
- **PROGRAM_ERROR** viene attivato quando PL/SQL riscontra dei problemi di esecuzione nel codice. SQLCODE viene impostato a -6501 (il codice di errore è ORA-06501).
- **STORAGE_ERROR** viene attivato quando PL/SQL necessita di più memoria di quella effettivamente disponibile, o quando individua un problema di corruzione dei dati. SQLCODE viene impostato su -6500 (il codice di errore è ORA-06500).
- **TIMEOUT_ON_RESOURCE** viene attivato quando una risorsa richiesta da ORACLE non è disponibile quando questo tenta di accedervi. Questa situazione indica generalmente un'istanza che possiede una terminazione non normale. SQLCODE viene impostato a -51 (il codice di errore è ORA-00051).
- **TOO_MANY_ROWS** viene attivato quando un'istruzione select (o una sotto-query) che dovrebbe restituire una sola riga ne restituisce più d'una. SQLCODE vale -1427 (il codice di errore è ORA-01427).
- **TRANSACTION_BACKED_OUT** viene attivato quando viene effettuato un rollback della parte remota di una transazione. SQLCODE vale -61 (il codice di errore è ORA-00061).

- **VALUE_ERROR** viene attivato quando il valore di una colonna o di una variabile PS/SQL risulta danneggiato (come nel caso di un troncamento). Questo tipo di problema avviene in genere durante la conversione tra tipi di variabili diversi, durante la copia di un valore da un campo all'altro, al termine di un calcolo numerico che provochi perdite di precisione nei risultati. Questo flag non viene attivato quando una stringa viene troncata durante il suo trasferimento in una variabile host (come nel caso di un **FETCH**). In questo caso l'indicatore della variabile host (se si è scelto di utilizzarne uno) viene impostato alla lunghezza corretta della stringa (prima del troncamento). Se la copia ha avuto successo, la variabile indicatrice vale 0. Vedere **VARIABILE INDICATORE. VALUE_ERROR** imposta **SQLCODE** su -6502 (il codice di errore è ORA-01476).
- **ZERO_DIVIDE** viene attivato quando un'istruzione cerca di dividere un numero per zero. **SQLCODE** vale -1476 (il codice di errore è ORA-01476).
- **OTHERS** è l'opzione che permette di gestire tutti i flag di eccezione non valutati nella sezione **EXCEPTION**. Deve sempre comparire come ultima istruzione e deve rimanere da sola. Non può accompagnarsi ad altre eccezioni.
- La gestione delle eccezioni, specie quando si tratta di eccezioni definite dall'utente, dovrebbe limitarsi agli errori fatali, ovvero quelli che provocano l'arresto del normale flusso dell'esecuzione.
- Se viene attivato un flag di eccezione che non risulta controllabile all'interno del blocco corrente, il programma salta alla sezione **EXCEPTION** del blocco a mano a mano più esterno, fino a che o l'eccezione viene valutata, 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. Questo significa che si possono gestire direttamente le variabili locali, mentre le altre devono essere precedute dal nome del blocco in cui sono state definite.
- Si presta particolare attenzione quando si controllano alcuni tipi particolari di eccezione. Ad esempio, se la sezione **EXCEPTION** prelevasse il messaggio di errore e lo inserisse all'interno di una tabella, l'eccezione **NOT_LOGGED_ON** entrerebbe in un ciclo infinito. Si progetti il codice 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 flusso dell'esecuzione passa automaticamente alla sezione di gestione delle eccezioni del blocco più esterno, o al programma principale. Vedere **RAISE** per ulteriori dettagli.

EXCEPTION_INIT

TIPO Funzione di PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE **DECLARE EXCEPTION, EXCEPTION, SQLCODE**

SINTASSI **PRAGMA EXCEPTION_INIT(eccezione, intero);**

DESCRIZIONE Le eccezioni di sistema standard, come **ZERO_DIVIDE**, gestite in base al loro nome non sono altro che associazioni di stringhe e codici di errore interni di ORACLE. Esistono centinaia di questi codici, ma solo ai più importanti sono stati assegnati dei nomi. Le eccezioni anonime attivano anch'esse dei flag e

trasferiscono il flusso dell'esecuzione al blocco EXCEPTION ma devono essere gestite nel sottoblocco OTHERS attraverso il loro codice di errore.

È comunque possibile assegnare dei nomi personalizzati ai codici di errore di ORACLE attraverso l'istruzione EXCEPTION_INIT. *eccezione* è il nome da assegnare al codice di errore fornito attraverso l'intero (vedere il manuale dei messaggi e dei codici di errore di ORACLE per un elenco completo). *intero* deve essere un numero negativo se lo è anche il codice di errore (cioè nel caso di errori fatali), mentre *eccezione* seguire le solite regole di nomenclatura.

Si noti 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, non è un vero e proprio comando eseguibile. Il pragma deve essere posto nella sezione DECLARE di un blocco PL/SQL e deve essere preceduto da una dichiarazione di eccezione.

ESEMPIO DECLARE

```
    un_errore      exception;
    pragma         exception_init(un_errore -666);
BEGIN
...
EXCEPTION
    when un_errore
        then ...
END;
```

EXECUTE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE PROCEDURE, CREATE FUNCTION, CREATE PACKAGE, CREATE PACKAGE BODY, GRANT (forma 2, privilegi degli oggetti), Capitolo 24 e Capitolo 25

SINTASSI execute *elemento_da_eseguire* [*argomenti*];

DESCRIZIONE Esegue una procedura, un package o una funzione. Per eseguire un procedura contenuta all'interno di un package, è necessario specificare sia il nome del package che quello della procedura sulla riga di comando, come risulta dall'esempio seguente. Questo esempio lancia una procedura di nome NEW_WORKER contenuta nel package chiamato LEDGER_PACKAGE; il valore 'ADAH TALBOT' viene passato come argomento alla procedura.

```
execute LEDGER_PACKAGE.NEW_WORKER('ADAH TALBOT');
```

Per lanciare un oggetto eseguibile, è necessario possedere il privilegio EXECUTE relativo all'oggetto in questione. Vedere GRANT (forma 2, privilegi degli oggetti).

EXECUTE (dinamico interno SQL)

TIPO Comando del precompilatore SQL

PRODOTTI Precompilatore

VEDERE ANCHE EXECUTE IMMEDIATE, PREPARE

SINTASSI EXEC SQL [FOR :*intero*]

```
    EXECUTE {nome_istruzione|nome_blocco}
    [USING :variabile[:indicatore] [, :variabile[:indicatore] ...]
```

DESCRIZIONE *intero* è una variabile host utilizzata per limitare il numero di iterazioni quando si utilizza l'opzione where su delle matrici. *nome_istruzione* rappresenta il nome di un'istruzione di aggiornamento dei dati (come insert, delete o update; l'uso di select non è consenito) mentre *nome_blocco* è il nome di un blocco PL/SQL. L'opzione USING introduce un elenco di sostituzioni a livello di variabili host da effettuare all'interno dell'istruzione da eseguire.

ESEMPIO

```
stringa_lavor : string(1..200);
    nome_lavor : string(1..25);
    get(stringa_lavor);
    exec sql at TALBOT prepare ADAH from :stringa_lavor;
    exec sql execute ADAH using :nome_lavor;
```

EXECUTE IMMEDIATE (dinamico interno SQL)

TIPO Comando del precompilatore SQL

PRODOTTI Precompilatore

VEDERE ANCHE EXECUTE, PREPARE

SINTASSI EXEC SQL [AT {database | :variabile_host}]
 EXECUTE IMMEDIATE {:stringa | letterale}

DESCRIZIONE *database* è il nome di una connessione a un database diverso da quello di default; *variabile_host* può contenere sia un nome che un valore. *stringa* è una stringa di una variabile dell'host che contiene un'istruzione SQL. *letterale* è una semplice stringa di caratteri contenente un'istruzione SQL. L'istruzione EXECUTE IMMEDIATE non può contenere riferimenti a variabili dell'host diverse da quelle specificate in *:stringa*. L'istruzione SQL viene interpretata, resa eseguibile, ed eseguita; al termine dell'esecuzione il codice eseguibile viene eliminato. 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 dell'interpretazione di ciascuna istruzione.

ESEMPIO

```
get(stringa_lavor);
    exec sql execute immediate :stringa_lavor;
    exec sql execute immediate
        "delete from SKILL where Skill = 'Grave Digger'";
```

EXISTS

TIPO Operatore SQL

PRODOTTI Tutti

VEDERE ANCHE ANY, ALL, IN, NOT EXISTS, Capitolo 11

SINTASSI select...
 where EXISTS (select...)

DESCRIZIONE EXISTS restituisce *true* se la sottoquery seguente restituisce almeno una riga. L'opzione select della sottoquery può essere una colonna, un valore letterale o un asterisco. Le convenzioni consigliano l'impiego di un asterisco o di una lettera 'x').

Molti programmatore preferiscono EXISTS ad ANY e ALL in quanto risulta più semplice da comprendere e impiegare. Inoltre, la maggior parte delle formule gestibili con ANY e ALL può essere ricostruita con degli EXISTS. Si noti, comunque, che sebbene ORACLE gestisca correttamente costrutti quali NOT EXISTS, > ALL e così

via, alcuni database SQL non lo fanno. Si faccia riferimento al libro *A Guide to the SQL Standard* di C. J. Date (1989, Addison-Wesley, N.Y.) per una trattazione più completa di questo argomento.

ESEMPIO La query riportata nell'esempio seguente utilizza EXISTS per elencare tutti i record della tabella COMPITO che hanno punteggi analoghi a quelli della tabella COMPITOLAVORATORE.

```
select COMPITO.Compito
  from COMPITO
 where EXISTS
   (select 'x' from COMPITOLAVORATORE
    where COMPITOLAVORATORE.Compito = COMPITO.Compito);
```

EXIT (forma 1, PL/SQL)

TIPO Funzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE END, LOOP, Capitolo 22

SINTASSI EXIT [*ciclo*] [WHEN *condizione*];

DESCRIZIONE Senza nessuna opzione, EXIT esce dal ciclo in corso e porta il flusso dell'esecuzione all'istruzione immediatamente successiva al ciclo. Se ci si trova all'interno di un ciclo annidato è possibile uscire da qualsiasi livello del ciclo specificando il nome di quest'ultimo. Quando si specifica una condizione, il ciclo viene interrotto quando questa risulta vera. All'uscita qualsiasi cursore eventualmente aperto all'interno del ciclo viene chiuso automaticamente.

ESEMPIO <>ada>>

```
for i in 1..100 loop
  ...
  <<giorgio>>
  for k in 1..100 loop
    ...
    exit when...
    delete...
    exit ada when...;
  end loop giorgio;
  ...
end loop ada;
```

EXIT (forma 2, SQL*PLUS)

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE COMMIT, DISCONNECT, QUIT, START

SINTASSI {EXIT | QUIT} [SUCCESS|FAILURE|WARNING|*intero*|*variabile*]

DESCRIZIONE EXIT termina una sessione SQL*PLUS e riporta l'utente al sistema operativo, al programma chiamante o al menu. EXIT salva eventuali modifiche pendenti nel database. Viene inoltre ritornato un codice d'errore. SUCCESS, FAILURE e WARNING hanno valori differenti da un sistema operativo all'altro; FAILURE e WARNING possono essere uguali su alcune piattaforme.

intero è il valore che si vuole passare al blocco chiamante come codice di ritorno; *variabile* permette di impostare questo valore in modo dinamico. Quest'ultima può essere una variabile di sistema (come sql.sqlcode che contiene sempre il codice sql dell'ultima istruzione SQL eseguita), una variabile definita dall'utente o un blocco SQL*PLUS o PL/SQL.

EXP

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE LN; NUMERI, FUNZIONI; Capitolo 7

SINTASSI EXP(*n*)

DESCRIZIONE EXP restituisce il valore neperiano *e* elevato alla sua potenza di ordine *n*; $e = 2.718281828\dots$

EXPLAIN PLAN

TIPO Procedura

PRODOTTI SQL*PLUS

VEDERE ANCHE Capitolo 36

SINTASSI EXPLAIN PLAN

```
[SET STATEMENT ID = nome] [INTO [utente.]tabella[@dblink]]  
FOR istruzione_sql
```

DESCRIZIONE *nome* rappresenta l'identificativo dell'*istruzione_sql* così come appare nella tabella di output; segue le normali convenzioni di nomenclatura. Se non viene indicato alcun nome, la colonna STATEMENT_ID della tabella risulta nulla. *tabella* rappresenta il nome del tabella di uscita in cui devono comparire le informazioni relative al piano. La tabella deve essere creata prima dell'esecuzione della procedura. Il file di partenza UTLXPLAN.SQL contiene il formato da utilizzare e può essere impiegato direttamente nella creazione della tabella di default (chiamata PLAN_TABLE). Se non viene specificato il parametro *tabella*, EXPLAIN PLAN utilizza PLAN_TABLE per default. *istruzione_sql* è un qualsiasi comando select, insert, update o delete di cui si voglia esaminare il piano di esecuzione di ORACLE.

EXPORT

Export ha due definizioni differenti.

- Export è l'utilità di ORACLE utilizzata per l'archiviazione di dati provenienti da database in formato ORACLE all'interno di file di esportazione utili per lo scambio di informazioni con altre tipologie di database. Questi file possono essere reimportati all'interno di ORACLE mediante il comando Import.
- L'esportazione è l'operazione che prevede l'utilizzo dell'utilità Export vista al punto precedente per riversare i dati in un file di formato differente.

Per informazioni riguardanti l'utilità Export vedere *Oracle Server Utilities User's Guide*.

FASE DI ESECUZIONE

La fase di esecuzione delle istruzioni SQL è quel lasso di tempo durante il quale vengono reperite tutte le informazioni necessarie per la loro esecuzione.

FEEDBACK (SQL*PLUS)

Vedere SET.

FETCH

Una delle fasi dell'esecuzione di una query è quella di fetch: le righe di dati che corrispondono ai criteri di ricerca vengono prelevate dal database.

FETCH (forma 1, interno SQL)

TIPO Comando Embedded SQL

PRODOTTI Precompilatori

VEDERE ANCHE CLOSE; DECLARE CURSOR; DESCRIBE; INDICATORE, VARIABILE; OPEN; PREPARE

SINTASSI EXEC SQL [FOR :*intero*] FETCH cursore
 INTO :variabile[[INDICATOR]:*indicatore*]
 [,:variabile[[INDICATOR]:*indicatore*]]...
 EXEC SQL [FOR :*intero*] FETCH cursore
 USING DESCRIPTOR *descrittore*

DESCRIZIONE *intero* è una variabile host che contiene il massimo numero di righe da inserire nelle variabili di output. *cursore* è il nome di un cursore impostato precedentemente da DECLARE CURSOR. *:variabile[:indicatore]* sono una o più variabili host in cui vengono posti i dati in uscita. Se anche solo una delle variabili host è un array, allora anche tutte le altre eventualmente presenti nell'elenco INTO devono essere matriciali. La parola chiave INDICATOR è opzionale.

descrittore è il descrittore di un precedente comando DESCRIBE.

ESEMPIO exec sql fetch CURSOR1 into :Nome,:Compito;

FETCH (forma 2, PL/SQL)

TIPO Istruzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE %FOUND, %ROWTYPE, CLOSE, DECLARE CURSOR, LOOP, OPEN, SELECT...INTO, Capitolo 22

SINTASSI FETCH *cursore* INTO {*record* | *variabile* [,*variabile*]...};

DESCRIZIONE FETCH preleva una riga di dati. Le istruzioni di selezione relative al cursore indicato determinano le colonne da prelevare mentre le relative clausole where determinano quali e quante righe prelevare. Questo blocco viene abitualmente chiamato "blocco attivo", ma non è disponibile all'utente fino a che non lo si preleva riga per riga attraverso l'istruzione FETCH.

L'istruzione FETCH preleva quindi una riga dal blocco attivo e inserisce i valori all'interno del record (o della stringa di variabili) definita da DECLARE.

Se si utilizza il metodo dell'elenco di variabili a ciascuna colonna del blocco attivo deve corrispondere una variabile, ciascuna 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 attraverso l'attributo %ROWTYPE in modo tale che la struttura del record sia la stessa di quella delle colonne del blocco attivo. Ogni variabile del record può quindi essere gestita individualmente utilizzando come prefisso il nome del record e come nome quello della colonna da cui proviene. Per ulteriori dettagli, si faccia riferimento al Capitolo 22.

ESEMPIO

```
declare
    pi      constant NUMBER(9,7) := 3.1415926;
    area   NUMBER(14,2);
    cursor rag_cursore is
        select * from VAL_RAGGI;
    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 a cui l'elemento figlio appartiene viene chiamato padre.

FILE

Vedere DATI, FILE.

FILE, TIPI

I file hanno in genere un nome e un'estensione. Ad esempio nel file comfort.tab, comfort rappresenta il nome vero e proprio mentre tab è l'estensione o "tipo" del file. In SQL*PLUS ai file di partenza creati con EDIT, se non viene specificata alcuna estensione, viene assegnata per default l'estensione SQL. In altre parole, il comando:

```
edit misti
```

crea o modifica un file di nome misti.sql, e:

```
start misti
```

cerca di eseguire 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 l'estensione di default dei file di spooling è LST. Naturalmente quando si specifica una qualsiasi estensione, sia edit che spool utilizzano quella fornita e non quella di default.

FILE DI CONTROLLO

Un file di controllo è un piccolo file amministrativo richiesto da ciascun database, necessario per avviare ed eseguire un sistema di database. Un file di controllo è abbinato a un database, non a un'istanza. Sono preferibili più file di controllo identici che un unico file.

FLOOR

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CEIL, FUNZIONI NUMERICHE, Capitolo 7

SINTASSI FLOOR(*valore*)

DESCRIZIONE FLOOR è il più grande intero minore o uguale al *valore* specificato.

ESEMPIO FLOOR(2)=2

FLOOR(1.3)=1

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 CICLI e CICLI A CURSORI.

FORMATO

Vedere BTITLE, COLONNA, DATA, TO_CHAR, TO_DATE, TTITLE.

FORMATO, MODELLO

Un modello di formato è un'opzione che controlla l'aspetto di numeri, date e stringhe di caratteri. I modelli di formato delle colonne di date vengono utilizzati all'interno delle funzioni di conversione TO_CHAR e TO_DATE.

FORMFEED

Formfeed è un carattere di controllo che segnala alla stampante di saltare all'inizio del foglio successivo.

FROM

TIPO Clausola SQL

PRODOTTI Tutti

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 link a un database remoto. Sia delete che select richiedono una clausola from che definisce 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 dell'utente.

Se la tabella è remota, è necessario definire preventivamente una connessione. In tal caso @*link* ne specifica il nome. Se si specifica la connessione ma non l'utente, la query cerca una tabella di proprietà dell'utente che ha eseguito il comando all'interno del database remoto. *condizione* può contenere una query correlata attraverso l'*alias* fornito.

FULL TABLE SCAN

Full table scan è un metodo di prelevamento dei dati in cui ORACLE cerca sequenzialmente i dati specificati all'interno di tutti i blocchi del database. Questo metodo si oppone quindi alla ricerca per indici. Vedere il Capitolo 36.

FUNZIONE

Una funzione è una operazione predefinita (come ad esempio “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 così via.

FUNZIONI DI ELENCO

TIPO Elenco

PRODOTTI Tutti

VEDERE ANCHE Tutte le varie funzioni di elenco, Capitolo 8

DESCRIZIONE Di seguito sono elencate tutte le funzioni di elenco SQL di ORACLE. Ciascuna di esse viene elencata con il suo nome in questo stesso glossario completo di sintassi ed esempi.

Questa funzione fornisce il valore massimo di un elenco:

GREATEST(*valore1*, *valore2*, ...)

Questa il valore minimo:

LEAST(*valore1*, *valore2*, ...)

FUNZIONI DI GRUPPO

TIPO Funzioni SQL

PRODOTTI Tutti e in particolare PL/SQL

VEDERE ANCHE AVG; COUNT; GLB; CLUB; MAX; MIN; NUMERI, FUNZIONI; STDDEV; SUM; VARIANZA; Capitolo 7

DESCRIZIONE Una funzione di gruppo calcola un singolo valore riassuntivo (ad esempio, una somma o una media) su un vettore di valori. Riportiamo un elenco alfabetico di tutte le funzioni di gruppo di SQL di ORACLE. Ognuna di queste viene analizzata in maggior dettaglio sotto il proprio nome. Le funzioni di gruppo risultano utili solo all'interno di query e sottoquery. DISTINCT consente di utilizzare le funzioni di gruppo per riassumere solo valori distinti.

NOME E SIGNIFICATO DELLE FUNZIONI

AVG([DISTINCT | ALL] valore) fornisce la media dei valori contenuti nel gruppo di righe indicato.

COUNT([DISTINCT | ALL] valore |*) fornisce il numero di righe di una colonna o di una tabella (quando si usa *).

GLB(etichetta) fornisce il massimo limite superiore di un'etichetta di sicurezza del sistema operativo.

LUB(etichetta) fornisce il minimo limite superiore di un'etichetta di sicurezza del sistema operativo.

MAX([DISTINCT | ALL] valore) fornisce il massimo dei valori contenuti in un gruppo di righe.

MIN([DISTINCT | ALL] valore) fornisce il minimo dei valori contenuti in un gruppo di righe.

STDDEV([DISTINCT | ALL] valore) fornisce la deviazione standard dei valori contenuti in un gruppo di righe.

SUM([DISTINCT | ALL] valore) fornisce la somma dei valori di un gruppo di righe.

VARIANCE([DISTINCT | ALL] valore) fornisce la varianza dei valori contenuti in un gruppo di righe.

FUZZY

La corrispondenza fuzzy è un metodo di ricerca che permette di ignorare eventuali errori di scrittura. Vedere OPERATORI DI RICERCA TESTUALE e il Capitolo 29.

GENITORE

In un albero, un genitore è un nodo che possiede un altro nodo sotto di lui (un figlio).

GESTORE DI BLOCCHI A LIVELLO DI RIGA

Si tratta di quella parte del kernel che permette di applicare blocchi alle righe piuttosto che alle tabelle, permettendo un più elevato livello di concorrenza e una migliore efficienza delle transazioni.

GET

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE EDIT, SAVE

SINTASSI GET *file* [LIST | NOLIST]

DESCRIZIONE GET preleva un file da un sistema di archiviazione remoto e lo inserisce nel buffer corrente (cioè 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 crei un elenco delle righe caricate (è il funzionamento di default). NO-LIST carica il file senza elencarne il contenuto.

ESEMPIO L'esempio seguente preleva il file di nome lavoro.sql:

```
get lavoro
```

GIST

Un gist è una versione ristretta di un'entità testuale. Un tipico esempio si ha nella gestione da parte di ConText. Vedere in proposito il Capitolo 30.

GLB

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE EDIT, SAVE

SINTASSI GLB *etichetta* [LIST | NOLIST]

DESCRIZIONE La funzione di gruppo GLB restituisce il limite inferiore maggiore di un'etichetta di sicurezza del sistema operativo. L'etichetta deve essere o un'espressione valutata in un valore MLSLABEL, o un testo letterale racchiuso tra virgolette nel formato di etichetta MLS standard. Questa funzione viene valutata solo in Trusted ORACLE; vedere *Trusted ORACLE Server Administrator's Guide* per ulteriori informazioni.

GOTO

TIPO Istruzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE BLOCCHI, STRUTTURA; Capitolo 22

SINTASSI GOTO *etichetta*;

DESCRIZIONE GOTO trasferisce il controllo alla sezione di codice preceduta dall'*etichetta*. Questa può precedere qualsiasi istruzione all'interno di un blocco dell'esecuzione o in una sezione EXCEPTION. Tuttavia, è necessario tener conto di alcune limitazioni intrinseche:

- GOTO non può trasferire l'esecuzione a un blocco interno al blocco corrente, a blocchi contenuti all'interno di cicli o di istruzioni IF.
- GOTO non può trasferire l'esecuzione a etichette esterne al proprio blocco a meno che non si tratti del blocco immediatamente esterno.

ESEMPIO Quello che segue è un ciclo formato da un GOTO. Conoscendo a priori qual è il numero massimo, questa routine 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 (forma 1, ruoli e privilegi di sistema)

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER USER, CREATE USER, PRIVILEGI, REVOKE, RUOLI, Capitolo 18

SINTASSI GRANT {privilegio_di_sistema | ruolo}
 [, {privilegio_di_sistema | ruolo}] ...
 TO {utente | ruolo | PUBLIC}[,{utente | ruolo | PUBLIC}]...
 [WITH ADMIN OPTION]

DESCRIZIONE 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 operazioni o istruzioni di controllo, come ALTER SYSTEM, CREATE ROLE o GRANT. Vedere PRIVILEGI per ulteriori dettagli sui privilegi. Gli utenti vengono creati attraverso CREATE USER e in seguito vengono concessi i privilegi di sistema che permettono loro la connessione al database e l'effettuazione di alcune operazioni. Un utente che non abbia alcun permesso non è in grado di fare alcunché all'interno di ORACLE. Un ruolo è una raccolta di privilegi creata attraverso il comando CREATE ROLE. Come accade per gli utenti, un ruolo quando viene creato non dispone di alcun privilegio. Ad entrambe le entità i privilegi vengono assegnati per mezzo di uno o più GRANT. È altresì possibile concedere dei ruoli ad altri ruoli, creando architetture annidate di permessi, e fornendo all'amministratore del sistema un'ampia libertà di scelta nella gestione della sicurezza del database.

L'opzione WITH ADMIN OPTION consente all'utente (o al ruolo) cui si sta concedendo il privilegio di rilasciare (o rimuovere) lo stesso privilegio ad altri utenti.

ESEMPIO Nel breve listato seguente viene concesso al ruolo BASE il permesso di collegarsi ad ORACLE, di modificare i parametri di sessione, di creare tabelle, viste e sinonimi. Inoltre viene consentito al ruolo il permesso di rilasciare gli stessi permessi ad altri ruoli o utenti.

```
grant CREATE SESSION, ALTER SESSION, CREATE TABLE,  

       CREATE VIEW, CREATE SYNONYM  

       to BASE  

       with admin option;
```

GRANT (forma 2, privilegi degli oggetti)

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE GRANT (Forma 1), PRIVILEGI, RUOLI, Capitolo 18

SINTASSI GRANT privilegio_dell'oggetto[,privilegio_dell'oggetto]...
 [(colonna [,colonna]...)]
 ON [utente.]oggetto
 TO {utente | ruolo | PUBLIC}[,{utente | ruolo | PUBLIC}]... }
 [WITH GRANT OPTION]

DESCRIZIONE Il secondo formato dell'istruzione GRANT concede privilegi relativi agli oggetti. Questi possono essere tabelle, viste, sequenze, procedure, funzioni, pacchetti, snapshot o sinonimi. La concessione di privilegi a sinonimi equivale alla concessione dei medesimi privilegi agli oggetti riferiti. Vedere la parte relativa ai PRIVILEGI per un elenco esaustivo di tutti i privilegi degli oggetti e del loro significato.

Quando si vogliono concedere privilegi INSERT, UPDATE e REFERENCES a tabella o viste, il comando GRANT può comprendere un elenco di colonne cui applicare i permessi. Quando invece GRANT viene utilizzato senza specificare alcuna colonna, i permessi vengono applicati a tutte le colonne.

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

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE COLLATION, LEAST, MAX, ALTRE FUNZIONI, Capitoli 7 e 8

SINTASSI GREATEST(*valore1*, *valore2*, ...)

DESCRIZIONE GREATEST sceglie il numero più grande da un elenco di valori. Questi possono essere colonne, valori letterali, espressioni o variabili di tipo CHAR, VARCHAR2, NUMBER e DATE. 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 "maggior". 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 una delle due, 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' viene considerato minore di '6' dato che 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 è necessario utilizzare prima la funzione TO_DATE sui letterali da valutare.

GREATEST_LB

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE LEAST_UB, FUNZIONI DI ELENCO

SINTASSI GREATEST_LB(*etichetta*[,*etichetta*]...)

DESCRIZIONE Restituisce il massimo limite inferiore di un elenco di etichette di Trusted ORACLE.

GROUP BY

TIPO Clausola SQL

PRODOTTI Tutti

VEDERE ANCHE HAVING, ORDER BY, WHERE, Capitolo 10

SINTASSI

```
SELECT espressione [, espressione]...
    GROUP BY espressione [, espressione]...
        HAVING condizione
        ...
    
```

DESCRIZIONE GROUP BY permette di produrre in abbinamento a select una riga di sommario per ciascuna riga che abbia valori identici all'interno di una o più espressioni definite dall'utente. Queste possono essere:

- costanti;
- funzioni senza parametri (SysDate, User);
- funzioni di gruppo quali SUM, AVG, MIN, MAX, COUNT;
- espressioni identiche a quelle specificate nella clausola GROUP BY.

Le colonne specificate nella clausola GROUP BY possono anche non essere ripetute nella clausola select; mentre devono comunque essere presenti all'interno della tabella.

Per determinare quali gruppi includere nella clausola GROUP BY è necessario utilizzare HAVING. L'opzione where, invece, determina le righe da includere nei gruppi.

GROUP BY e HAVING seguono WHERE, CONNECT BY e START WITH. La clausola ORDER BY viene eseguita dopo WHERE, GROUP BY e HAVING (le quali a loro volta vengono eseguite in quell'ordine)..

Nella clausola ORDER BY è possibile specificare una funzione di gruppo e la colonna su cui eseguirla (anche quando non hanno nulla a che vedere con le funzioni di gruppo e le colonne delle clausole SELECT, GROUP BY e HAVING. D'altra parte, se si specifica una colonna dell'ORDER BY che non appartiene a una funzione di gruppo, comunque deve essere contenuta all'interno della clausola GROUP BY.

ESEMPIO

```
select Persona, COUNT(Articolo), SUM(Importo) Totale
      from REGISTRO
      where Azione = 'PAGATO'
      group by Persona
      having COUNT(Articolo) > 1
      order by AVG(Importo);
```

GRUPPO DI AGGIORNAMENTO DI SNAPSHOT

Si tratta di un insieme di snapshot locali il cui aggiornamento avviene contemporaneamente. Questi gruppi permettono di mantenere la coerenza dei dati tra snapshot diversi. Vedere il Capitolo 28.

HAVING

Vedere GROUP BY.

HEADING (SQL*PLUS)

Vedere SET.

HEADSEP (SQL*PLUS)

Vedere SET.

HELP

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

SINTASSI HELP {*argomento* | COMMANDS | CLAUSES}
HELP

DESCRIZIONE Se è disponibile un file di guida per l'argomento indicato, SQL*PLUS raccoglie tutte le informazioni inerenti e le visualizza. HELP COMMANDS mostra un elenco di tutti i comandi per cui è disponibile la guida di aiuto. HELP CLAUSES visualizza un elenco di clausole e di argomenti correlati. HELP da solo mostra gli argomenti per cui sono disponibili i file di aiuto. La tabella di HELP può essere creata e caricata durante la creazione del database.

HEXTORAW

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE RAWTOHEX

SINTASSI HEXTORAW(*stringa_esadecimale*)

DESCRIZIONE HEXTORAW trasforma una stringa di cifre esadecimali in notazione binaria.

HINT

All'interno di una query è possibile specificare degli hint che dirigano gli ottimizzatori di costi. Per specificare uno di questi hint si utilizzi la sintassi seguente: si digiti la stringa che segue immediatamente dopo la parola chiave select:

/*+

Quindi si aggiunga un hint, ad esempio:

FULL(lavoratore)

Si termini l'hint con la sequenza di caratteri seguente:

*/

Vedere il Capitolo 36 per una descrizione di tutti gli hint disponibili e del loro impatto sulla gestione delle query.

HOST (SQL*PLUS)

TIPO SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE \$, @, @@, AVVIO

SINTASSI HO[ST] *host comando*

DESCRIZIONE Un host è un computer su cui è attivo ORACLE RDBMS. Il comando HOST di SQL*PLUS passa qualsiasi comando dell'host al sistema operativo in modo che possa essere eseguito senza uscire da SQL*PLUS. SQL*PLUS permette di includere nella stringa di comando eventuali variabili locali o PL/SQL.

Di questa funzione purtroppo non è garantito il funzionamento su tutte le possibili piattaforme e sistemi operativi.

HTF

HTF è una libreria di funzioni ipertestuali, richiamabile dalle procedure. La libreria HTF è stata progettata per essere utilizzata con l'ORACLE Web Application Server. Le tabelle seguenti riassumono le componenti di questa libreria; per una trattazione esaustiva dell'ORACLE Web Application Server, vedere la guida utente *ORACLE Web Application Server User's Guide*.

TAG DI STRUTTURA

| FUNZIONE | ATTRIBUTI |
|-----------|---|
| htmlOpen | Nessun attributo |
| htmlClose | Nessun attributo |
| headOpen | Nessun attributo |
| headClose | Nessun attributo |
| bodyOpen | (cbackground in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| bodyClose | Nessun attributo |

TAG DEGLI ELEMENTI HEAD

| FUNZIONE | ATTRIBUTI |
|----------|--|
| title | (ctitle in varchar2) restituisce varchar2 |
| htitle | (ctitle in varchar2 nsize in integer DEFAULT 1 callign in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| base | Nessun attributo |
| isindex | (prompt in varchar2 DEFAULT NULL curl in varchar2 DEFAULT NULL) restituisce varchar2 |
| linkRel | (rel in varchar2 curl in varchar2 ctitle in varchar2 DEFAULT NULL) restituisce varchar2 |
| linkRev | (rev in varchar2 curl in varchar2 ctitle in varchar2 DEFAULT NULL) restituisce varchar2 |
| meta | (http_equiv in varchar2 cname in varchar2 content in varchar2) restituisce varchar2 |
| nextid | (identifier in varchar2) restituisce varchar2 |

TAG DEGLI ELEMENTI BODY

| FUNZIONE | ATTRIBUTI |
|--------------------------|--|
| hr | (cclear in varchar2 DEFAULT NULL csrc in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) return varchar2 |
| iTag degli elementi BODY | Funzione Attributi |

| TAG DEGLI ELEMENTI BODY | |
|--------------------------------|--|
| FUNZIONE | ATTRIBUTI |
| ine | (cclear in varchar2 DEFAULT NULL csrc in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| br | (cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| nl | (cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| header | (nsize in integer cheader in varchar2 calign in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| anchor | (curl in varchar2 ctext in varchar2 cname in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| mailto | (caddress in varchar2 ctext in varchar2 cname in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| img | (curl in varchar2 calign in varchar2 DEFAULT NULL calt in varchar2 DEFAULT NULL cismap in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| paragraph | (calign in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| address | (cvalue in varchar2 cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| comment | (ctext in varchar2) restituisce varchar2 |
| preOpen | (cclear in varchar2 DEFAULT NULL cwidth in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| blockquoteOpen | (cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| TAG DEGLI ELEMENTI LIST | |
| FUNZIONE | ATTRIBUTI |
| listHeader | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |

TAG DEGLI ELEMENTI LIST**FUNZIONE****ATTRIBUTI**

| | |
|----------|--|
| listItem | (ctext in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cdingbat in varchar2 DEFAULT NULL csrc in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|----------|--|

| | |
|-----------|--|
| ulistOpen | (cclear in varchar2 DEFAULT NULL cwrap in varchar2 DEFAULT NULL cdingbat in varchar2 DEFAULT NULL csrc in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|-----------|--|

| | |
|-----------|--|
| olistOpen | (cclear in varchar2 DEFAULT NULL cwrap in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|-----------|--|

| | |
|-----------|--|
| dlistOpen | (cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|-----------|--|

| | |
|-----------|--|
| dlistTerm | (ctext in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|-----------|--|

| | |
|----------|--|
| dlistDef | (ctext in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|----------|--|

ELEMENTI SEMANTICI**FUNZIONE****ATTRIBUTI**

| | |
|------|--|
| cite | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|------|--|

| | |
|------|--|
| code | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|------|--|

| | |
|----|--|
| em | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|----|--|

| | |
|----------|--|
| emphasis | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|----------|--|

| | |
|----------|--|
| keyboard | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|----------|--|

| | |
|-----|--|
| kbd | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|-----|--|

| | |
|--------|--|
| sample | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|--------|--|

| | |
|--------|--|
| strong | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|--------|--|

| | |
|----------|--|
| variable | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|----------|--|

TAG DI FORMATTAZIONE**FUNZIONE****ATTRIBUTI**

| | |
|------|--|
| bold | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
|------|--|

| TAG DI FORMATTAZIONE | |
|-----------------------------|--|
| FUNZIONE | ATTRIBUTI |
| italic | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| teletype | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| MODULI WEB | |
| FUNZIONE | ATTRIBUTI |
| formOpen | (curl in varchar2 cmethod in varchar2 DEFAULT 'POST') restituisce varchar2 |
| formCheckbox | (cname in varchar2 cvalue in varchar2 DEFAULT 'on' cchecked in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| formHidden | (cname in varchar2 cvalue in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| formImage | (cname in varchar2 csrc in varchar2 calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| formPassword | (cname in varchar2 csize in varchar2 DEFAULT NULL cmaxlength in varchar2 DEFAULT NULL cvalue in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| formRadio | (cname in varchar2 cvalue in varchar2 cchecked in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| formReset | (cvalue in varchar2 DEFAULT 'Reset' cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| formSubmit | (cname in varchar2 DEFAULT NULL cvalue in varchar2 DEFAULT 'Submit' cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| formText | (cname in varchar2 csize in varchar2 DEFAULT NULL cmaxlength in varchar2 DEFAULT NULL cvalue in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| formSelectOpen | (cname in varchar2 cprompt in varchar2 DEFAULT NULL nsize in integer DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| formSelectOption | (cvalue in varchar2 cselected in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |

| | |
|------------------|--|
| formTextarea | (cname in varchar2 nrows in integer ncolumns in integer calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| formTextareaOpen | (cname in varchar2 nrows in integer ncolumns in integer calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |

| TAG DI TABELLE FUNZIONE | ATTRIBUTI |
|----------------------------|---|
| tableOpen | (cborder in varchar2 DEFAULT NULL calign in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| tableCaption | (ccaption in varchar2 calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| tableRowOpen | (calign in varchar2 DEFAULT NULL cvalign in varchar2 DEFAULT NULL cdp in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| tableHeader | (cvalue in varchar2 DEFAULT NULL calign in varchar2 DEFAULT NULL cdp in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL crowspan in varchar2 DEFAULT NULL ccolspan in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |
| tableData | (cvalue in varchar2 DEFAULT NULL calign in varchar2 DEFAULT NULL cdp in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL crowspan in varchar2 DEFAULT NULL ccolspan in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) restituisce varchar2 |

| FUNZIONI SPECIALI FUNZIONE | ATTRIBUTI |
|-------------------------------|--|
| escape_sc | (ctext in varchar2) restituisce varchar2 |

| FUNZIONI COSTANTI FUNZIONE | VALORE |
|-------------------------------|--|
| para | constant varchar2(3) := '<P>' |
| preClose | constant varchar2(6) := '</PRE>' |
| blockquoteClose | constant varchar2(13) := '</BLOCKQUOTE>' |
| ulistClose | constant varchar2(5) := '' |

| FUNZIONI COSTANTI | |
|--------------------------|--|
| FUNZIONE | VALORE |
| olistClose | constant varchar2(5) := '' |
| dlistClose | constant varchar2(5) := '</DL>' |
| menuListOpen | constant varchar2(6) := '<MENU>' |
| menuListClose | constant varchar2(7) := '</MENU>' |
| dirListOpen | constant varchar2(5) := '<DIR>' |
| dirListClose | constant varchar2(6) := '</DIR>' |
| formSelectClose | constant varchar2(9) := '</SELECT>' |
| formTextareaClose | constant varchar2(11) := '</TEXTAREA>' |
| formClose | constant varchar2(7) := '</FORM>' |
| tableRowClose | constant varchar2(5) := '</TR>' |
| tableClose | constant varchar2(8) := '</TABLE>' |

HTP

HTP è una libreria di procedure ipertestuali richiamabili da procedure SQL. A ogni chiamata HTP corrisponde una chiamata HTF; a meno che non si stiano annidando diverse chiamate HTF all'interno di una sola chiamata HTP è bene utilizzare HTP. La libreria HTP è progettata per lavorare insieme a ORACLE Web Application Server. Le tabelle seguenti riassumono le componenti della libreria HTP; per informazioni dettagliate su ORACLE Web Application Server, si consulti la guida utente *ORACLE Web Application Server User's Guide*.

| TAG DI STRUTTURA | |
|-------------------------|--|
| PROCEDURA | ATTRIBUTI |
| htmlOpen | Nessun attributo |
| htmlClose | Nessun attributo |
| headOpen | Nessun attributo |
| headClose | Nessun attributo |
| bodyOpen | (cbackground in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| bodyClose | Nessun attributo |

| TAG ELEMENTO HEAD | |
|--------------------------|--|
| PROCEDURA | ATTRIBUTI |
| htitle | (ctitle in varchar2) |
| htitle | (ctitle in varchar2 nsize in integer DEFAULT 1 calign in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |

| TAG ELEMENTO HEAD | |
|--------------------------|---|
| PROCEDURA | ATTRIBUTI |
| base | Nessun attributo |
| isindex | (cprompt in varchar2 DEFAULT NULL curl in varchar2 DEFAULT NULL) |
| linkRel | (crel in varchar2 curl in varchar2 ctitle in varchar2 DEFAULT NULL) |
| linkRev | (rev in varchar2 curl in varchar2 ctitle in varchar2 DEFAULT NULL) |
| meta | (charset in varchar2 name in varchar2 content in varchar2) |
| nextid | (identifier in varchar2) |

| TAG ELEMENTO BODY | |
|--------------------------|--|
| PROCEDURA | ATTRIBUTI |
| hr | (cclear in varchar2 DEFAULT NULL csrc in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| line | (cclear in varchar2 DEFAULT NULL csrc in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| br | (cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| nl | (cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| header | (nsize in integer header in varchar2 calign in varchar2 DEFAULT NULL nowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| anchor | (url in varchar2 ctext in varchar2 name in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| mailto | (address in varchar2 ctext in varchar2 name in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| img | (url in varchar2 calign in varchar2 DEFAULT NULL alt in varchar2 DEFAULT NULL cismap in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| para | Nessun attributo |

| TAG ELEMENTO BODY | |
|--------------------------|---|
| PROCEDURA | ATTRIBUTI |
| paragraph | (calign in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| address | (cvalue in varchar2 cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| comment | (ctext in varchar2) |
| preOpen | (cclear in varchar2 DEFAULT NULL cwidth in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| preClose | Nessun attributo |
| blockquoteOpen | (cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| blockquoteClose | Nessun attributo |
| TAG ELEMENTO LIST | |
| PROCEDURA | ATTRIBUTI |
| listHeader | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) |
| listItem | (ctext in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cdingbat in varchar2 DEFAULT NULL csrc in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| ulistOpen | (cclear in varchar2 DEFAULT NULL cwrap in varchar2 DEFAULT NULL cdingbat in varchar2 DEFAULT NULL csrc in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| ulistClose | Nessun attributo |
| olistOpen | (cclear in varchar2 DEFAULT NULL cwrap in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| olistClose | Nessun attributo |
| dlistOpen | (cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| dlistTerm | (ctext in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| dlistDef | (ctext in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| dlistClose | Nessun attributo |

TAG ELEMENTO LIST

| PROCEDURA | ATTRIBUTI |
|------------------|------------------|
| menulistOpen | Nessun attributo |
| menulistClose | Nessun attributo |
| dirlistOpen | Nessun attributo |
| dirlistClose | Nessun attributo |

ELEMENTI SEMANTICI

| PROCEDURA | ATTRIBUTI |
|------------------|---|
| cite | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) |
| code | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) |
| em | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) |
| emphasis | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) |
| keyboard | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) |
| kbd | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) |
| sample | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) |
| strong | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) |
| variable | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) |

TAG DI FORMATTAZIONE

| PROCEDURA | ATTRIBUTI |
|------------------|---|
| bold | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) |
| italic | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) |
| teletype | (ctext in varchar2 cattributes in varchar2 DEFAULT NULL) |

MODULI WEB

| PROCEDURA | ATTRIBUTI |
|------------------|---|
| formOpen | (curl in varchar2 cmethod in varchar2 DEFAULT 'POST') |
| formCheckbox | (cname in varchar2 cvalue in varchar2 DEFAULT 'on' cchecked in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |

| MODULI WEB PROCEDURA | ATTRIBUTI |
|-------------------------|---|
| formHidden | (cname in varchar2 cvalue in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| formImage | (cname in varchar2 csrc in varchar2 calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| formPassword | (cname in varchar2 csize in varchar2 DEFAULT NULL cmaxlength in varchar2 DEFAULT NULL cvalue in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| formRadio | (cname in varchar2 cvalue in varchar2 cchecked in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| formReset | (cvalue in varchar2 DEFAULT 'Reset' cattributes in varchar2 DEFAULT NULL) |
| formSubmit | (cname in varchar2 DEFAULT NULL cvalue in varchar2 DEFAULT 'Submit' cattributes in varchar2 DEFAULT NULL) |
| formText | (cname in varchar2 csize in varchar2 DEFAULT NULL cmaxlength in varchar2 DEFAULT NULL cvalue in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| formSelectOpen | (cname in varchar2 cprompt in varchar2 DEFAULT NULL nsize in integer DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| formSelectOption | (cvalue in varchar2 cselected in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| formSelectClose | No attributi |
| formTextarea | (cname in varchar2 nrows in integer ncolumns in integer calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| formTextareaOpen | (cname in varchar2 nrows in integer ncolumns in integer calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| formTextareaClose | Nessun attributo |
| formClose | Nessun attributo |

| TABELLE PROCEDURA | ATTRIBUTI |
|--|--|
| tableOpen | (cborder in varchar2 DEFAULT NULL calign in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| tableCaption | (ccaption in varchar2 calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| tableRowOpen | (calign in varchar2 DEFAULT NULL cvalign in varchar2 DEFAULT NULL cdp in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL catttributes in varchar2 DEFAULT NULL) |
| tableHeader | (cvalue in varchar2 DEFAULT NULL calign in varchar2 DEFAULT NULL cdp in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL crowspan in varchar2 DEFAULT NULL ccolspan in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| tableData | (cvalue in varchar2 DEFAULT NULL calign in varchar2 DEFAULT NULL cdp in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL crowspan in varchar2 DEFAULT NULL ccolspan in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL) |
| tableRowClose | Nessun attributo |
| tableClose | Nessun attributo |
| PROCEDURE DI OUTPUT PROCEDURA | ATTRIBUTI |
| print | (cbuf in varchar2 DEFAULT NULL) |
| print | (dbuf in date) |
| print | (nbuf in number) |
| prn | (cbuf in varchar2 DEFAULT NULL) |
| prn | (dbuf in date) |
| prn | (nbuf in number) |
| p | (cbuf in varchar2 DEFAULT NULL) |
| p | (dbuf in date) |
| p | (nbuf in number) |
| prints | (ctext in varchar2) |
| ps | (ctext in varchar2) |
| escape_sc | (ctext in varchar2) |

NOTA La procedura *P* è una chiamata abbreviata alla procedura *PRINT*. La procedura *PRN* serve per stampare gli avanzamenti riga.

IF

TIPO Istruzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE CICLO, Capitolo 22

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 la *condizione* vale TRUE, quindi il programma salta alla riga che segue END IF. Se la *condizione* vale FALSE, vengono valutate tutte le condizioni dei vari ELSIF. 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 è vera vengono eseguite le istruzioni che seguono l'istruzione ELSE opzionale. Si noti che ELSE non ha alcuna condizione associata.

ESEMPIO declare

```
pi      constant NUMBER(9,7) := 3.1415926;
area   NUMBER(14,2);
cursor rag_cursore is
    select * from RADIUS_VALS;
val_rag rag_cursore%ROWTYPE;
begin
    open rag_cursore;
    fetch rag_cursore into rad_val;
    area := pi*power(rad_val.raggio,2);
    if area >30
    then
        insert into AREA values (rad_val.raggio, area);
    end if;
    close rag_cursore;
end;
.
/

```

IMPORT

Import è l'utilità di ORACLE che serve a recuperare i dati da file di formati compatibili con altri database. Vedere EXPORT. Per dettagli sull'uso di Import vedere la guida utente *ORACLE Server Utilities Users Guide*.

IN

TIPO Operatore logico della clausola where

PRODOTTI Tutti

VEDERE ANCHE ALL, ANY, OPERATORI LOGICI, Capitolo 3

SINTASSI where *espressione* IN ({'stringa' [, 'stringa']... | *select...*})

DESCRIZIONE IN è equivalente a =ANY. Nella prima forma consente di confrontare l'uguaglianza dell'espressione rispetto a tutte le eventuali stringhe dell'elenco. Nel secondo caso vengono confrontati i valori delle righe selezionate dalla sottoquery. Si tratta di due costrutti logicamente equivalenti, ma che lavorano su dati diversi. IN funziona, oltre che con i tipi RowID, anche con i tipi di dati VARCHAR2, CHAR, DATE e NUMBER.

INDICATORE, VARIABLE

TIPO Campo di dati

PRODOTTI Precompilatori PL/SQL e ORACLE

VEDERE ANCHE : (due punti, il prefisso delle variabili host)

SINTASSI *:nome[INDICATOR]:indicatore*

DESCRIZIONE Una variabile indicatore, utilizzata quando sia il nome che l'indicatore sono nomi di variabili del linguaggio host, è un campo di dati il cui valore indica se considerare o meno una variabile host equivalente a NULL (la parola chiave INDICATOR è fornita per leggibilità e non ha alcuna funzione specifica).

nome può essere una qualsiasi variabile host contenuta nella sezione delle dichiarazioni del precompilatore. *indicatore* viene definito anch'esso nella sezione delle dichiarazioni del precompilatore ed equivale a un intero di due byte.

Alcuni linguaggi procedurali consentono l'utilizzo diretto di variabili contenenti NULL o valori indefiniti. ORACLE e SQL richiedono invece alcune minime modifiche. Per far sì che i linguaggi di ORACLE risultino compatibili con le variabili NULL si imposta una variabile indicatore alla variabile host in modo che si comporti come un flag, indicando cioè se la variabile è pari a NULL oppure no. L'accoppiata variabile dell'host e variabile indicatore, sebbene possa essere sciolta nel linguaggio host, viene sempre gestita da SQL e PL/SQL come un'unica entità.

ESEMPIO BEGIN

```
select Nome, Compito into :Lavoratore, :JobCompito:JobCompitoInd
    from COMPITOLAVORATORE
  where Nome = :Primo||'':Cognome;

  IF :JobCompito:JobCompitoInd IS NULL
  THEN :JobCompito:JobCompitoInd := 'Nessun compito'
  END IF;
END;
```

Si noti che sia la verifica del contenuto della variabile che la conseguente assegnazione della stringa 'Nessun compito' se il contenuto precedente era NULL vengono effettuate utilizzando il nome di variabile concatenata.

PL/SQL è sufficientemente intelligente da controllare la variabile indicatore nel primo caso e impostare il valore della variabile host nel secondo. L'insieme delle due variabili viene trattato esattamente come una colonna di ORACLE. È bene fare attenzione ai seguenti aspetti.

- All'interno di qualsiasi blocco PL/SQL le variabili host devono sempre essere utilizzate nello stesso modo (ovvero sempre da sole, oppure sempre concatenate agli indicatori).
- Un blocco PL/SQL non può riferirsi direttamente a una variabile indicatore (mentre può farlo il programma host).

Quando si imposta il valore delle variabili host da un programma host (ma al di fuori di PL/SQL) occorre tenere presenti le seguenti implicazioni.

- 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 che nelle operazioni di insert e update.
- Se si imposta il valore della variabile indicatore a un valore maggiore o uguale a 0, la variabile concatenata viene considerata come contenente lo stesso valore della variabile host.
- PL/SQL controlla il valore di tutte le variabili indicatore all'ingresso di un blocco, reimpostandole all'uscita.

Quando si caricano dei valori nelle variabili concatenate dall'interno di PL/SQL attraverso 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 della 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.

INDICE

Indice è un termine generale che identifica una caratteristica di ORACLE/SQL utilizzata generalmente per velocizzare l'esecuzione e imporre l'unicità di alcuni dati. Gli indici forniscono un metodo di accesso rapido ai dati delle tabelle durante l'esecuzione di scansioni complete. Esistono diversi tipi di indici; *vedere* INDICE CONCATENATO, INDICE COMPRESO e INDICE UNICO.

INDICE COMPRESO

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" (troncate) e non registrate per ridurre lo spazio richiesto dall'indice.

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 suddivide in partizioni una tabella, i dati in essa contenuti vengono archiviati all'interno di nuove tabelle. All'atto della creazione di un indice relativo alla tabella suddivisa, si può scegliere di suddividere anche quest'ultimo in modo da rispecchiare la struttura delle nuove partizioni. Quando le partizioni degli indici non coincidono con quelle delle tabelle, l'indice si dice *globale*. Vedere il Capitolo 17.

INDICE LOCALE

Quando viene partizionata una tabella, i suoi dati vengono archiviati all'interno di tabelle diverse. Quando viene creato un indice a partire dalla tabella partizionata, questo può essere partizionato a sua volta per rispecchiare l'aspetto della partizione. Queste partizioni indicizzate sono dette *indici locali*. Vedere il Capitolo 17.

INDICE UNICO

Un indice unico è un indice che impone l'unicità di ogni valore che indirizza. L'indice può essere un'unica colonna o un tipo di dati concatenato (formato di diverse colonne). Vedere VINCOLO DI INTEGRITÀ.

INDICI, SEGMENTO

Il segmento di un indice è lo spazio allocato per gli indici di una tabella, in contrapposizione a quello allocato per i dati.

init.ora

init.ora è un file di configurazione del database che contiene impostazioni e nomi di file utilizzati all'avvio del sistema mediante il comando CREATE DATABASE o i comandi di avvio e di uscita. Vedere l'*ORACLE Server Administrator's Guide*.

INITCAP

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; LOWER; UPPER; Capitolo 6

SINTASSI INITCAP(*stringa*)

DESCRIZIONE INITCAP rende maiuscola la prima lettera di una o più parole. Tiene conto della eventuale presenza di spazi e simboli (virgole, punti, ;, !, @, #, \$ e così via) e opera le modifiche sulle lettere specificate di seguito.

ESEMPIO INITCAP('ecco.qui,un-esempio di!come@initcap#funziona')

produce la stringa seguente:

Ecco.Qui,Un-Esempio Di!Come@Initcap#Funziona

INPUT

TIPO Comando dell'editor della riga di comando di SQL*PLUS

PRODOTTI Tutti

VEDERE ANCHE APPEND, CHANGE, DEL, EDIT, Capitolo 5

SINTASSI I [INPUT] [*testo*]

DESCRIZIONE INPUT aggiunge una nuova riga di testo alla fine del buffer corrente. INPUT senza parametri permette di introdurre diverse linee di testo, interrompendosi quando riceve un carattere di INVIO su una riga vuota. Lo spazio tra INPUT e *testo* non viene aggiunto alla riga. Ogni eventuale altro spazio viene aggiunto. Vedere DEL per una discussione relativa alla riga corrente.

INSERT (forma 1, interno SQL)

TIPO Comando interno di SQL

PRODOTTI Precompilatori

VEDERE ANCHE EXECUTE, FOR

SINTASSI EXEC SQL [AT{*database*}:*variabile_host*] [FOR :*intero*]
 INSERT INTO [*utente*.]*tabella*[@*db_link*] [(*colonna* [,*colonna*]...)]
 { VALUES (*espressione* [,*espressione*]...) | *query* }

DESCRIZIONE *database* è il nome di un database diverso da quello di default mentre *variabile_host* contiene il nome del database. :*intero* è una variabile 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_collegamento* è il nome del database remoto in cui è stata archiviata la tabella. Vedere INSERT, forma 3, per una discussione sulla clausola VALUES, sull'utilizzo delle colonne e di query.

espressione può essere sia un'espressione vera e propria sia una variabile host nella forma :*variabile*[:*indicatore*].

INSERT (forma 2, PL/SQL)

TIPO Istruzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE CURSOR SQL, Capitolo 22

SINTASSI INSERT INTO [*utente*.]*tabella*[@*db_link*] [(*colonna* [,*colonna*]...)]
 VALUES (*espressione* [,*espressione*]...) | *query*...);

DESCRIZIONE La modalità di utilizzo di INSERT in PL/SQL è identica al suo formato generale (il formato 3 del paragrafo seguente) salvo le eccezioni riportate di seguito.

- È possibile utilizzare variabili PL/SQL nelle espressioni dell'elenco VALUES.
- Ogni variabile viene trattata allo stesso modo di una costante (così come lo sarebbe il suo valore esplicito).
- Se si utilizza la versione *query*... di INSERT, non è consentita la clausola INTO di select.

INSERT (forma 3, comando SQL)

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE TABLE, Capitolo 4, Capitolo 25 e Capitolo 26

SINTASSI INSERT INTO [*utente*.]*tabella*[@*db_link*] [(*colonna* [,*colonna*]...)]
 { VALUES (*espressione* [,*espressione*]...) | *query* }

DESCRIZIONE INSERT aggiunge una o più righe alla tabella o alla vista indicata. Il campo *utente*, opzionale, deve possedere il privilegio per l'inserimento. *tabella* è la

tabella in cui inserire le righe. Se viene fornito un elenco di colonne, per ciascuna deve essere verificata un'espressione SQL. Tutte le colonne non presenti in elenco vengono riempite con il valore NULL e nessuna di esse dev'essere stata definita NOT NULL altrimenti INSERT fallisce. Se non viene fornito alcun elenco di colonne, devono essere forniti i valori per tutte le colonne.

INSERT assieme a una query aggiunge, colonna per colonna, il numero di righe risultanti dalla query. Se non viene fornito alcun elenco di colonne, si deve garantire che le tabelle abbiano lo stesso numero e tipo di colonne. Un esempio viene fornito dal comando DESCRIBE di SQL*PLUS.

ESEMPIO La riga seguente inserisce una riga nella tabella COMFORT:

```
insert into COMFORT values ('KEENE','23-SEP-93',99.8,82.6,NULL);
```

Questa riga invece inserisce Citta, DataCampione (entrambe colonne NOT NULL) e Precipitazione all'interno di COMFORT:

```
insert into COMFORT (Citta, DataCampione, Precipitazione) values
('KEENE','22-DEC-93',3.9);
```

Per copiare solo i dati della Citta di KEENE in una nuova tabella di nome NEW_HAMPSHIRE si può utilizzare:

```
insert into NEW_HAMPSHIRE select * from COMFORT
where Citta = 'KEENE';
```

Vedere il Capitolo 4 e il Capitolo 25 per informazioni riguardanti l'inserimento di righe all'interno di tabelle che utilizzano tipi di dati astratti. Vedere il Capitolo 26 per dettagli sull'inserimento di righe all'interno di tabelle annidate e array variabili.

INTEGRITÀ REFERENZIALE

L'integrità referenziale è la proprietà che garantisce che i valori di una colonna dipendano da quelli presenti in un'altra colonna. Questa proprietà viene mantenuta attraverso regole di integrità. Vedere VINCOLO DI INTEGRITÀ.

INSTEAD OF, TRIGGER

Il trigger INSTEAD OF esegue un blocco di codice PL/SQL al posto della transazione che provoca l'esecuzione del trigger. Questi tipi di trigger vengono comunemente impiegati nel reindirizzamento delle transazioni dirette a viste. Vedere CREATE TRIGGER, il Capitolo 23 e il Capitolo 25.

INSTR

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; SUBSTR; Capitolo 6

SINTASSI INSTR(*stringa*,*set*[,*inizio*[,*ricorrenza*]])

DESCRIZIONE INSTR trova la posizione di un *set* di caratteri in una *stringa*, partendo dalla posizione di *inizio* della stringa e cercando nella prima, seconda, terza... *ricorrenza* del *set*. Questa funzione lavora con i tipi NUMBER e DATE. *inizio* può essere un numero negativo, in tal caso la ricerca comincia dalla fine della stringa e procede all'indietro.

ESEMPIO Per trovare la terza ricorrenza di 'PI' nella stringa seguente si può impiegare INSTR:

```
INSTR('PETER PIPER PICKED A PECK OF PICKLED PEPPERS','PI',1,3)
```

Il risultato di questa funzione è 30, posizione della terza ricorrenza di 'PI'.

INSTRB

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; INSTR; Capitolo 6

SINTASSI INSTRB(*stringa, set[,inizio[,ricorrenza]]*)

DESCRIZIONE INSTRB restituisce la posizione di un *set* di caratteri all'interno di una *stringa*, partendo dalla sua posizione di *inizio* e cercando la prima, la seconda... *ricorrenza* del set. Anche questa funzione lavora con i tipi NUMBER e DATE. *inizio* può essere negativo, nel qual caso la ricerca comincia dalla fine della stringa e procede all'indietro.

Questa funzione è analoga a INSTR per serie di caratteri singoli.

INTERSECT

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE MINUS, OPERATORI DI QUERY, UNION, Capitolo 11

SINTASSI select...

```
    INTERSECT  
        select...
```

DESCRIZIONE INTERSECT combina due query e restituisce solo le righe della prima selezione che siano identiche ad almeno una riga di quelle della seconda istruzione select. Il numero di colonne e di tipi di dati dev'essere uguale tra i due comandi select ad eccezione dei nomi delle colonne. Vedere il Capitolo 11 per una trattazione sulle importanti differenze tra INTERSECT, UNION e MINUS.

INTESTAZIONE DI RIGA

È quella parte della riga che contiene le meta-informationi, cioè il numero di campi, i tipi delle colonne e così via.

IS NULL

TIPO Operatore logico

PRODOTTI Tutti

VEDERE ANCHE OPERATORI LOGICI, Capitoli 3 e 7

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 qualcos'altro, ivi compreso un altro valore NULL. Per una disquisizione sul problema del NULL vedere il Capitolo 7.

JOIN

TIPO Definizione

PRODOTTI Tutti

VEDERE ANCHE SELECT, Capitolo 3

SINTASSI WHERE {tabella.colonna = tabella.colonna}

DESCRIZIONE Un'operazione di join combina colonne e dati da due o più tabelle (e in alcuni rari casi da parti diverse della stessa tabella). Le tabelle sono elencate nella clausola from dell'istruzione select e la relazione tra le due tabelle è specificata nella clausola where generalmente attraverso una semplice equazione del tipo:

```
where LAVORATORE.Alloggio = ALLOGGIO.Alloggio
```

Questo costrutto viene spesso indicato col nome di *equi-join* a causa dell'uso del segno di uguale nella clausola where. Si possono congiungere delle tabelle anche utilizzando altre forme relazionali come \geq , $<$ e così via, ottenendo raramente risultati significativi. Molto spesso uno o entrambi i membri della clausola possono contenere delle espressioni: talvolta delle combinazioni di colonne o i risultati di una SUBSTR. Il join di due tabelle senza alcuna clausola where produce un semplice prodotto cartesiano, in cui ciascuna riga di una tabella si combina con tutte le righe dell'altra. Una tabella di 80 righe combinata con un'altra di 100 righe produce un risultato di 8000 righe (in genere ben poco significativo).

Il *join esterno* è un metodo che permette di recuperare le righe di una tabella che non combaciano con quelle di un'altra tabella. Un classico esempio è fornito dalla tabella dei lavoratori e dei loro compiti. Si supponga di aver combinato la tabella LAVORATORE con quella COMPITO tramite un equi-join:

```
select LAVORATORE.Nome, Compito
      from LAVORATORE, COMPITOLAVORATORE
     where LAVORATORE.Nome = COMPITOLAVORATORE.Nome
```

Sfortunatamente si ottengono solo i lavoratori che hanno un compito (cioè quelli che hanno una riga nella tabella COMPITOLAVORATORE). Come si possono allora elencare tutti i lavoratori e i loro compiti senza boicottare quelli privi di compito? Un join esterno mediante un segno + permette di risolvere il problema:

```
select LAVORATORE.Nome, Compito
      from LAVORATORE, COMPITOLAVORATORE
     where LAVORATORE.Nome = COMPITOLAVORATORE.Nome(+);
```

Il segno più (+) sta a significare “se esiste una riga nella tabella LAVORATORE a cui non corrisponde alcuna riga della tabella COMPITOLAVORATORE, la si aggiunga ugualmente”. In effetti è come aggiungere una riga NULL, ma permette di visualizzare tra i risultati tutte le righe di LAVORATORE comprese quelle che non hanno una corrispondenza in COMPITOLAVORATORE. Vedere il Capitolo 11 per una discussione completa.

ISTANZA

Un'istanza è tutto quel che viene richiesto da ORACLE per la sua esecuzione: 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 il mezzo attraverso cui si può 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 di particolari intervalli di valori. Vedere ANALYZE.

ISTRUZIONE

Un'istruzione SQL è un comando di ORACLE.

ISTRUZIONE SQL ESEGUIBILE

È un'istruzione eseguibile SQL che genera una chiamata al database. Di questa categoria fanno parte quasi tutte le query, le istruzioni DML, DDL e DCL.

KERNEL

Il kernel è il programma di base di ORACLE RDBMS (in realtà è una raccolta di diversi moduli) richiamabile dai processi in background.

LABEL (PL/SQL)

Un'etichetta è una parola associata a un'istruzione eseguibile, in genere per funziona da obiettivo per una istruzione GOTO.

LAST_DAY

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE ADD_MONTHS; DATA, FUNZIONI; NEXT_DAY; Capitolo 8

SINTASSI LAST_DAY(*data*)

DESCRIZIONE LAST_DAY fornisce la data dell'ultimo giorno del mese a cui appartiene la data indicata.

ESEMPIO Questa riga:

```
LAST_DAY ('05-NOV-98')
```

produce come risultato 30-NOV-98.

LEAST

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE GREATEST, FUNZIONI DI ELENCO, Capitolo 8

SINTASSI LEAST(*valore1, valore2, ...*)

DESCRIZIONE LEAST è il minimo 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-89') se non con l'ausilio della funzione TO_DATE. Vedere GREATEST per una discussione relativa alla valutazione dei numeri negativi.

LEAST_UB**TIPO** Funzione SQL**PRODOTTI** Tutti**VEDERE ANCHE** GREATEST_LB, FUNZIONI DI ELENCO**SINTASSI** LEAST_UB(*etichetta*[,*etichetta*]...)**DESCRIZIONE** Restituisce l'estremo superiore di un elenco di etichette di Trusted ORACLE. Le etichette vengono utilizzate nell'assegnazione di livelli di sicurezza ai record protetti. Per informazioni sussidiarie *vedere la Trusted ORACLE Server Administrator's Guide*.**LENGTH****TIPO** Funzione SQL**PRODOTTI** Tutti**VEDERE ANCHE** CARATTERI, FUNZIONI; VSIZE; Capitolo 6**SINTASSI** LENGTH(*stringa*)**DESCRIZIONE** LENGTH restituisce la lunghezza di una stringa, di un numero, di una data o di una espressione.**LENGTHB****TIPO** Funzione SQL**PRODOTTI** Tutti**VEDERE ANCHE** CARATTERI, FUNZIONI; LENGTH; VSIZE; Capitolo 6**SINTASSI** LENGTHB(*stringa*)**DESCRIZIONE** LENGTHB fornisce la lunghezza di una stringa, di un numero, una data o una espressione in byte (ottuple di bit) piuttosto che in caratteri. In questo modo è possibile determinare l'occupazione in byte piuttosto che in caratteri.**LETTURA RIPETIBILE**

La lettura ripetibile è una funzionalità per cui query successive restituiscono un insieme di risultati coerente in quanto l'aggiornamento ai dati viene ritardato fino all'esecuzione di tutte le query.

LEVEL**TIPO** Pseudocolonna**PRODOTTI** Tutti**VEDERE ANCHE** CONNECT BY, PSEUDOCOLONNE, Capitolo 12**SINTASSI** LEVEL

DESCRIZIONE Level è una pseudocolonna, utilizzata con CONNECT BY, il cui valore è 1 nel nodo radice, 2 per un figlio di primo livello, 3 per un figlio di secondo livello e così via. Level fornisce un'indicazione della profondità del nodo nell'albero. Il Capitolo 11 contiene una discussione estesa di CONNECT BY e del concetto di livello.

LGWR

LGWR (LoG WRiter process) scrive i log della SGA nei redo log. *Vedere PROCESSI IN BACKGROUND*.

LIBRARY

Come avveniva già in ORACLE8, i blocchi di codice PL/SQL possono accedere a sottoprogrammi esterni archiviati all'interno di librerie. Vedere CREATE LIBRARY.

LIKE

TIPO Comando SQL

PRODOTTI Tutti

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 cifre, spazi o caratteri. Se LIKE ha un _ o un % nella prima posizione di una stringa di confronto (come nel secondo degli esempi forniti) tutti gli indici della colonna vengono ignorati.

ESEMPIO 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 due 0."

LIMITAZIONE

Una regola di limitazione concernente una parte di dati (quale 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

TIPO Comando dell'editor della riga di comando di SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE APPEND, CHANGE, EDIT, INPUT, RUN, Capitolo 5

SINTASSI L[IST] [{*inizio**} [*fine**]]

DESCRIZIONE LIST elenca le righe del buffer corrente a partire da *inizio* fino a *fine*. Gli estremi sono numeri interi. La riga finale diventa la riga corrente del buffer e viene evidenziata con un asterisco. LIST senza parametri elenca tutte le righe. Un asterisco in una delle due posizioni sostituisce l'indicazione della riga corrente. LIST con un solo parametro elenca solo la riga richiesta; LIST * visualizza la sola riga corrente. Lo spazio tra LIST e *inizio* non è necessario ma aiuta la leggibilità.

ESEMPIO Per visualizzare il buffer corrente di SQL:

```
list
```

```
1 select Persona, Importo
2   from REGISTRO
3  where Importo > 10000
4*   and Tasso = 3;
```

L'asterisco mostra che la riga corrente è la numero 4.

Per visualizzare la sola riga 2 si digiti:

LIST 2

```
2* from REGISTRO
```

Questa chiamata ha anche l'effetto di rendere corrente la riga 2.

LN

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE NUMERI, FUNZIONI; Capitolo 7

SINTASSI $\text{LN}(\text{numero})$

DESCRIZIONE LN è il logaritmo naturale (o neperiano, o in base e) di un *numero*.

LOB

Un LOB è un oggetto molto grande. ORACLE supporta diversi tipi di dati di grandi dimensioni, ivi compresi i BLOB (binary large object), i CLOB (character large object) e i BFILE (binary file, archiviati all'esterno del database). Vedere il Capitolo 27 e la clausola LOB di CREATE TABLE.

LOCK TABLE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE COMMIT, DELETE, INSERT, ROLLBACK, SAVEPOINT, UPDATE

SINTASSI `LOCK TABLE [utente.]tabella[@db_link]
[, [utente.]tabella[@db_link]]...
IN modalità MODE [NOWAIT];`

DESCRIZIONE LOCK TABLE blocca una tabella utilizzando modalità che ne permettono la condivisione ma senza compromissione dell'integrità dei dati. L'uso di LOCK TABLE permette di fornire ad altri utenti l'accesso parziale e continuativo alla tabella. A prescindere dall'opzione scelta, la tabella resta nella modalità richiesta fino a che non si effettui un commit o un rollback sulle proprie transazioni.

Le modalità di blocco comprendono ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE, SHARE, SHARE ROW EXCLUSIVE ed EXCLUSIVE.

I blocchi di tipo EXCLUSIVE permettono agli utenti la sola query dei dati della tabella bloccata e null'altro. 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 blocco 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 può accedere alla tabella nello stesso istante).

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.

LOG

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE NUMERI, FUNZIONI; Capitolo 7

SINTASSI LOG(*base*, *numero*)

DESCRIZIONE LOG fornisce il logaritmo in base 10 del numero indicato.

ESEMPIO LOG(*base*, *valore*)

```
LOG(EXP(1),3) = 1.098612 // log(e) di 3  
LOG(10,100)   = 2       // log(10) di 100
```

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. Questo sistema di accesso in genere è distinto da quello del sistema operativo ospite.

LOGON ACCOUNT

Un logon account equivale a un LOGIN ACCOUNT.

LONG (SQL*PLUS)

Vedere SET.

LONG, TIPO DI DATI

Vedere TIPI DI DATI.

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 grezzi. I valori devono essere inseriti in notazione esadecimale.

LOWER

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; INITCAP; UPPER; Capitolo 6

SINTASSI LOWER(*stringa*)

DESCRIZIONE LOWER converte in minuscola qualsiasi lettera alfabetica della stringa fornita.

ESEMPIO LOWER('PENINSULA') = peninsula

LPAD

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; LTRIM; RPAD; RTRIM; Capitolo 6

SINTASSI LPAD(*stringa*,*lunghezza* [, 'set'])

DESCRIZIONE LPAD rende una stringa di una ben determinata *lunghezza* aggiungendo un certo *set* di caratteri a sinistra della stringa. Se non viene specificato alcun *set*, il carattere di riempimento è lo spazio.

ESEMPIO LPAD('>',11,'-')

produce

- - - - >

LTRIM

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; LPAD; RPAD; RTRIM; Capitolo 6

SINTASSI LTRIM(*stringa* [, 'set'])

DESCRIZIONE LTRIM taglia tutte le occorrenze (anche parziali) di un *set* di caratteri in una stringa da sinistra a destra.

ESEMPIO LTRIM('NANCY','AN')

produce:

CY

LUB

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE COMPUTE, GLB, FUNZIONI DI GRUPPO, Capitolo 7

SINTASSI LUB(*etichetta*)

DESCRIZIONE LUB fornisce il minimo estremo superiore di un'etichetta di sicurezza del sistema operativo. L'espressione deve essere di tipo MLSLABEL o essere un letterale racchiuso tra apici. Il valore risultante è di tipo RAW MLSLABEL. Questa funzione è disponibile solo all'interno di Trusted ORACLE. Vedere la guida *Trusted ORACLE Administrator's Guide* per ulteriori informazioni.

MAKE_REF

La funzione MAKE_REF costruisce un riferimento tra la chiave esterna di una tabella e la vista di un oggetto. MAKE_REF permette di sovrapporre riferimenti a relazioni di chiave esterna preesistenti. Vedere il Capitolo 31.

MAX

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE COMPUTE, FUNZIONI DI GRUPPO, MIN, Capitolo 7

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.

MAXDATA (SQL*PLUS)

Vedere SET.

MECCANISMI DI BLOCCO DEI FILE

I meccanismi di blocco dei file sono uno dei tre tipi di protezione interna. Queste protezioni assicurano che i file (di dati, di controllo e di redo log) siano letti e aggiornati correttamente.

METODO

Un metodo è un blocco di codice PL/SQL utilizzato per incapsulare i metodi di accesso ai dati di ciascun oggetto. All'interno di ORACLE i metodi sono specificati come parti dei tipi astratti (*vedere CREATE TYPE*) e il loro corpi sono dichiarati attraverso il comando CREATE TYPE BODY. Gli utenti possono eseguire i metodi sui tipi di dati per cui sono stati definiti. Il metodo costruttore creato per ciascun tipo di dati astratto è un esempio di metodo. *Vedere* il Capitolo 25 per ulteriori esempi di metodi.

MIN

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE COMPUTE, FUNZIONI DI GRUPPO, MAX, Capitolo 7

SINTASSI MIN([DISTINCT | ALL] *valore*)

DESCRIZIONE MIN restituisce il minimo dei 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 equivalente al minimo di tutti i valori presi distintamente.

MINUS

TIPO Funzione SQL, funzione di ricerca testuale

PRODOTTI Tutti

VEDERE ANCHE INTERSECT, OPERATORI DI QUERY, UNION, OPERATORI DI RICERCA TESTUALE, Capitoli 11 e 29

SINTASSI select
 MINUS
 select

All'interno di query ConText:

```
select colonna
      from TABELLA
     where CONTAINS(Text,'text MINUS text') >0;
```

DESCRIZIONE MINUS combina due query. Restituisce solo le righe del primo select che non sono state prodotte anche dal secondo (il risultato del primo MENO il risultato del secondo). Il numero di colonne e i tipi di dati devono essere identici, condizione non richiesta per i nomi delle colonne. *Vedere* il Capitolo 11 per una discussione sulle importanti differenze di INTERSECT, MINUS e UNION.

MOD

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE NUMERI, FUNZIONI; Capitolo 7

SINTASSI MOD(*valore*, *divisore*)

DESCRIZIONE MOD effettua una divisione intera tra un *valore* e 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. Il divisore non deve mai essere 0.

| | | |
|---------------|---------------|---------|
| ESEMPI | MOD(100,10) | = 0 |
| | MOD(22,23) | = 22 |
| | MOD(10,3) | = 1 |
| | MOD(-30.23,7) | = -2.23 |
| | MOD(4.1,.3) | = .2 |

Il secondo esempio mostra il comportamento di MOD qualora il divisore sia più grande del dividendo. Il risultato è il dividendo stesso. Si noti anche il caso importante:

$$\text{MOD}(\text{valore}, 1) = 0$$

è vera se *valore* è un numero intero. Questo è un ottimo trucco per verificare che un numero sia effettivamente un intero.

MODALITÀ ESCLUSIVA

La modalità di accesso esclusivo è una limitazione di accesso; viene posta su quelle risorse che per loro natura non consentono accessi contemporanei; il proprietario della limitazione assume i diritti esclusivi sulla modifica della risorsa in questione.

MONTAGGIO DI UN DATABASE

Il montaggio di un database equivale a renderlo disponibile all'amministratore del sistema.

MONTAGGIO E APERTURA DI UN DATABASE

Il montaggio e l'apertura di un database consentono di renderlo disponibile per gli utenti.

MONTHS_BETWEEN

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE ADD_MONTHS; DATA, FUNZIONI; Capitolo 8

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.

MULTIPROCESSO

Si tratta di un metodo di gestione dei database che consente la condivisione tra vari utenti dei dati contenuti al loro interno pur mantenendo una garanzia di integrità dei dati.

NEAR

TIPO Operatore di ricerca testuale

PRODOTTI ConText

VEDERE ANCHE CONTAINS, OPERATORI DI RICERCA TESTUALE, Capitolo 29

DESCRIZIONE All'interno di ConText, NEAR indica che per le stringhe specificate deve essere effettuata una ricerca di prossimità. Se i termini da cercare fossero ‘estate’ e ‘vacanza’ una ricerca di prossimità potrebbe essere strutturata nella maniera seguente:

```
select Text  
      from SONNET  
     where CONTAINS(Text,'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.

NLSSORT

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE *ORACLE Server Administrator's Guide*, Capitolo 32

SINTASSI NLSSORT(*carattere*)

DESCRIZIONE NLSSORT fornisce il valore (intero) del codice di caratteri NLS sulla base dell'opzione National Language Support selezionata per l'ambiente operativo in uso.

NLS_INITCAP

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; INITCAP; Capitolo 7

SINTASSI NLS_INITCAP(*numero*[, *parametri_nls*])

DESCRIZIONE NLS_INITCAP è analoga alla funzione INITCAP tranne per il fatto che gestisce una stringa di parametri racchiusa tra apici singoli. Fornisce un metodo per rendere maiuscole sequenze di caratteri particolari di alcune lingue internazionali.

NLS_LOWER

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; LOWER; Capitolo 7

SINTASSI `NLS_LOWER(numero[, parametri_nls])`

DESCRIZIONE NLS_LOWER è analoga alla funzione LOWER salvo per il fatto che consente la gestione di una stringa di parametri. Questi, racchiusi tra apici singoli, forniscono un metodo per rendere minuscole delle sequenze di caratteri particolari di alcuni linguaggi internazionali.

NLS_UPPER

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; UPPER; Capitolo 7

SINTASSI `NLS_UPPER(numero[, parametri_nls])`

DESCRIZIONE NLS_UPPER è analoga alla funzione UPPER salvo per il fatto che consente la gestione di una stringa di parametri. Questa, racchiusa tra apici singoli, fornisce un metodo per rendere maiuscole alcune sequenze di caratteri particolari di alcuni linguaggi internazionali.

NOAUDIT (forma 1, oggetti schema)

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE AUDIT, CREATE DATABASE LINK, VISTE DEL DIZIONARIO DI DATI, Capitolo 32

SINTASSI `NOAUDIT { opzione [,opzione]... | ALL }`
`ON {utente.oggetto}`
`[WHENEVER [NOT] SUCCESSFUL]`

DESCRIZIONE Questa forma di NOAUDIT interrompe l'ascolto di un'opzione per l'uso di una tabella, di una vista o di un sinonimo. Per far ciò è necessario esserne i proprietari 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'ascolto. Per le tabelle le opzioni sono ALTER, AUDIT, COMMENT, DELETE, GRANT, INDEX, INSERT, LOCK, RENAME, SELECT e UPDATE. Per procedure, funzioni, tipi, librerie e pacchetti possono essere ascoltati i comandi EXECUTE, per gli snapshot i comandi SELECT, per le directory i READ. GRANT ascolta sia i comandi GRANT che REVOKE. NOAUDIT GRANT li interrompe entrambi; ALL li interrompe tutti.

ON *oggetto* fornisce un nome agli oggetti in corso di ascolto (comprese tabelle, viste, sinonimi, sequenze).

Non è possibile utilizzare ALTER e INDEX per le viste. Le opzioni di default delle viste vengono create a partire dall'unione delle opzioni delle tabelle relative e delle 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'ascolto degli aggiornamenti delle tabelle che hanno avuto successo. WHENEVER NOT SUCCESSFUL disattiva l'ascolto degli

aggiornamenti non conclusi. Se si omette questa clausola opzionale entrambi i tipi di ascolto vengono disattivati.

Entrambe le forme concludono qualsiasi operazione aperta sul database. Se le tabelle remote vengono interrogate attraverso una connessione le auscultazioni del sistema remoto si basano sulle opzioni impostate in locale. Le informazioni relative all'ascolto vengono riportate nella tabella SYS.AUD\$. Vedere il Capitolo 32 per dettagli sulle viste di auditing.

ESEMPI La riga seguente interrompe l'ascolto di tutti i tentativi di aggiornamento e cancellazione della tabella LAVORATORE:

```
noaudit update, delete on LAVORATORE;
```

Questa riga interrompe l'ascolto di tutti gli accessi alla tabella LAVORATORE che non abbiano avuto successo:

```
noaudit all on LAVORATORE whenever not successful;
```

NOAUDIT (forma 2, istruzioni SQL)

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE AUDIT, NOAUDIT (forma 1), PRIVILEGI

SINTASSI NOAUDIT {*istruzione* | *privilegio_di_sistema*}
[,{*istruzione* | *privilegio_di_sistema*}]...
[BY *utente*[,*utente*]...]
[WHENEVER [NOT] SUCCESSFUL]

DESCRIZIONE NOAUDIT interrompe l'ascolto delle istruzioni SQL tenute sotto controllo a seguito del comando AUDIT (Formato 2). Interrompe sia le istruzioni semplici sia quelle autorizzate da privilegi di sistema (vedere AUDIT e PRIVILEGI). Se è presente la clausola BY e una lista di utenti, il comando interrompe l'ascolto delle istruzioni da essi impartite. In caso contrario ORACLE interrompe l'ascolto delle istruzioni impartite da tutti gli utenti. L'opzione WHENEVER SUCCESSFUL interrompe il controllo sulle sole istruzioni completate con successo; WHENEVER NOT SUCCESSFUL disabilita l'ascolto delle istruzioni che generano errori.

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 a cui viene collegato ciascun 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

L'opzione NOLOGGING nella creazione di oggetti (tabelle, LOB, indici, partizioni e così via) impone che l'operazione non venga trascritta nei file di log; per questo motivo non sarà ripristinabile a seguito di un guasto del sistema di archiviazione. Non

viene registrata neanche alcuna transazione diretta all'oggetto. NOLOGGING rim-piazza la parola chiave UNRECOVERABLE di ORACLE7.2.

NOMI DEGLI OGGETTI

Gli oggetti di un database devono necessariamente possedere un nome. Questo è vero per tabelle, viste, sinonimi, alias, colonne, indici, utenti, tablespace e così via. Esistono alcune regole che vincolano la nomenclatura degli oggetti di ORACLE.

- Il nome di un oggetto può avere da 1 a 30 caratteri di lunghezza, con l'eccezione dei nomi di database che sono limitati a 8 caratteri e ai nomi dei file host che spesso sono di 8 caratteri più un'estensione. I nomi degli snapshot non possono superare 19 caratteri di lunghezza così come il nome della tabella di base di un log di snapshot.
- I nomi non possono contenere virgolette.
- I nomi devono:
 - cominciare con una lettera;
 - contenere i soli caratteri A-Z, 0-9, \$, # e _;
 - non essere parole chiave riservate di ORACLE (*vedere PAROLE RISERVATE*).
 - non duplicare il nome di un altro oggetto del database appartenente allo stesso utente.

Lettere maiuscole o minuscole non fanno differenza nei nomi degli oggetti. I nomi dovrebbero essere significativi e utilizzare per quanto possibile una convenzione di nomenclatura come quella vista nel Capitolo 2 e analizzata in maggior dettaglio nel Capitolo 35.

NON-EQUI-JOIN

Un non-equi-join è una condizione di join diversa da quella di uguaglianza (=). *Vedere EQUI-JOIN.*

NOT

TIPO Operatore SQL

PRODOTTI Tutti

VEDERE ANCHE OPERATORI LOGICI, Capitolo 3

DESCRIZIONE NOT precede e complementa l'effetto di ciascuno dei seguenti operatori logici: BETWEEN, IN, LIKE ed EXISTS. NOT può anche precedere NULL come in IS NOT NULL.

NOT EXISTS

TIPO Operatore SQL

PRODOTTI Tutti

VEDERE ANCHE ANY, ALL, EXISTS, IN, Capitolo 11

SINTASSI select ...

 where NOT EXISTS (select...);

DESCRIZIONE NOT EXISTS restituisce un valore booleano all'interno di un'espressione where. Se la sottoquery che la segue restituisce una o più righe il valore boole-

ano è falso. La condizione di selezione della sottoquery può essere una colonna, un letterale o un asterisco.

ESEMPIO NOT EXISTS viene utilizzato spesso per determinare quali record di una tabella non abbiano record corrispondenti in un'altra. La query riportata di seguito usa NOT EXISTS per escludere dalla tabella COMPITO tutti quei record che non hanno capacità corrispondenti nella tabella COMPITOLAVORATORE. Il risultato di questa query è un elenco di tutti i compiti non coperti da alcun dipendente.

```
select COMPITO.Compito
  from COMPITO
 where NOT EXISTS
   (select 'x' from COMPITOLAVORATORE
    where COMPITOLAVORATORE.Compito = COMPITO.Compito);

COMPITO
-----
SCAVATORE
```

NULL (forma 1, PL/SQL)

TIPO Istruzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE BLOCCO, STRUTTURA

SINTASSI NULL;

DESCRIZIONE L'istruzione NULL non ha nulla a che vedere con il valore NULL. Il suo scopo principale è quello di rendere alcuni blocchi di codice più leggibili dicendo loro letteralmente di "non fare nulla". È anche un espediente per creare blocchi nulli (dato che PL/SQL vuole almeno un'istruzione tra BEGIN ed END). In genere viene utilizzato a seguito di una sequenza 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)

TIPO Valore di colonna SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE TABLE, FUNZIONI DI GRUPPO, VARIABILE INDICATORE, NVL, Capitolo 3

DESCRIZIONE Un valore NULL è un valore sconosciuto, irrilevante o non significativo. Tutti i tipi di dati ORACLE possono valere NULL. Questo significa che qualsiasi colonna ORACLE può possedere su una certa riga un valore nullo (a meno che la tabella non sia stata creata specificando l'opzione NOT NULL su quella colonna). NULL nei tipi di dati numerici non è la stessa cosa di zero.

Sono pochi i linguaggi procedurali che supportano direttamente l'idea di una variabile contenente un valore sconosciuto o NULL. ORACLE e SQL operano questo

supporto in maniera molto intuitiva. Per estendere i linguaggi di cui ORACLE ha sviluppato i precompilatori al concetto di variabile NULL si ricorre al concetto di 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 indicatrice possono essere utilizzate e impostate separatamente nel linguaggio host ma risultano sempre concatenate sia in SQL che in PL/SQL.

La funzione NVL può servire a rilevare l'assenza di valori su una colonna e a convertire le celle ad un valore significativo del tipo della colonna. Ad esempio, NVL(Nome,'SCONOSCIUTO') converte un valore NULL nella colonna Nome nella stringa "SCONOSCIUTO". Nel caso di valori non-NNULL (ad esempio quando è presente un nome) la funzione NVL restituisce semplicemente il valore della cella. NVL lavora in maniera analoga con numeri e date.

A 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 che IS NULL e IS NOT NULL) non funzionano con valori NULL. Ad esempio:

```
where Name = NULL
```

non è una clausola where valida. NULL richiede necessariamente la parola chiave IS:

```
where Name IS NULL
```

Durante gli ordinamenti di tipo order by, i valori NULL precedono tutti gli altri nei sort ascendenti e vengono per ultimi in quelli discendenti.

Quando vengono archiviati dei valori nulli in un database, essi vengono rappresentati attraverso un unico byte se ricadono tra due colonne di valori reali ma non vengono rappresentati se cadono al fondo di una riga (oltre il margine di colonna definito da create table). Se alcune colonne 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, tranne che nel caso in cui tutti i valori di un cluster chiave siano NULL.

NUMERI, FUNZIONI

Di seguito è riportato un elenco alfabetico di tutte le funzioni numeriche supportate dall'SQL di ORACLE. Ciascuna di queste funzioni viene trattata in maggior dettaglio all'interno di questo stesso capitolo assieme alla sua sintassi e a esempi di utilizzo. Queste funzioni possono essere utilizzate sia in SQL standard che in PL/SQL, tranne VSIZE che non è disponibile in PL/SQL.

| FUNZIONE | SIGNIFICATO |
|-------------------|-------------|
| valore1 + valore2 | Somma |
| valore1 - valore2 | Differenza |

| FUNZIONE | SIGNIFICATO |
|--|--|
| <i>valore1</i> * <i>valore2</i> | Prodotto |
| <i>valore1</i> / <i>valore2</i> | Divisione |
| ABS(<i>valore</i>) | Valore assoluto di <i>valore</i> |
| CEIL(<i>valore</i>) | Tetto di <i>valore</i> |
| COS(<i>valore</i>) | Coseno di <i>valore</i> |
| COSH(<i>valore</i>) | Coseno iperbolico di <i>valore</i> |
| EXP(<i>valore</i>) | e (numero neperiano) elevato alla <i>valoresima</i> potenza |
| FLOOR(<i>valore</i>) | Base di <i>valore</i> |
| LN(<i>valore</i>) | Logaritmo naturale (in base e) di <i>valore</i> |
| LOG(<i>base</i> , <i>valore</i>) | Logaritmo in base <i>base</i> di <i>valore</i> |
| MOD(<i>valore</i> , <i>divisore</i>) | Resto della divisione intera fra <i>valore</i> e <i>divisore</i> |
| NVL(<i>valore</i> , <i>sostituto</i>) | Sostituisce il <i>valore</i> quando vale NULL |
| POWER(<i>valore</i> , <i>esponente</i>) | Eleva <i>valore</i> all' <i>esponente</i> indicato |
| ROUND(<i>valore</i> , <i>precisione</i>) | Arrotonda il <i>valore</i> alla precisione desiderata |
| SIGN(<i>valore</i>) | Vale 1 se il <i>valore</i> è positivo, -1 se è negativo, 0 se è zero |
| SIN(<i>valore</i>) | Seno di <i>valore</i> |
| SINH(<i>valore</i>) | Seno iperbolico di <i>valore</i> |
| SQRT(<i>valore</i>) | Radice quadrata di <i>valore</i> |
| TAN(<i>valore</i>) | Tangente di <i>valore</i> |
| TANH(<i>valore</i>) | Tangente iperbolica di <i>valore</i> |
| TRUNC(<i>valore</i> , <i>precisione</i>) | Valore troncato alla precisione desiderata |
| VSIZE(<i>valore</i>) | Occupazione in byte del <i>valore</i> in ORACLE |

NUMERICO, TIPO DI DATI

Un tipo numerico 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

TIPO Opzioni di comandi SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE COLUMN, Capitolo 13

DESCRIZIONE Queste opzioni funzionano sia con il comando set numformat che con column format.

| FORMATO | DEFINIZIONE |
|---------|---|
| 9999990 | Il numero di nove o zeri determina il massimo numero di cifre visualizzabili. |

| FORMATO | DEFINIZIONE |
|----------------|---|
| 999,999,999.99 | I punti e le cifre decimali vengono piazzati all'interno del modello indicato. Se il valore è zero non visualizza nulla. |
| 999990 | Visualizza zero se il valore è zero. |
| 099999 | Visualizza i numeri anteponendo degli zeri. |
| \$99999 | Pone il simbolo valutario del dollaro prima di ogni numero. |
| B99999 | Se il valore è zero non visualizza nulla. È l'impostazione di default. |
| 99999MI | Se il valore è negativo, il segno meno seguirà il numero. L'impostazione di default vuole il segno meno a sinistra. |
| 99999S | Identico a 99999MI. |
| S99999 | Se il valore è negativo, il segno meno precede il numero, se il valore è positivo, un segno più precede il numero. |
| 99D99 | Visualizza il carattere decimale nella posizione indicata. |
| C99999 | Visualizza il simbolo valutario ISO nella posizione indicata. |
| L99999 | Visualizza il simbolo valutario locale nella posizione indicata. |
| RN | Visualizza il valore in numeri romani. |
| 99999PR | I valori negativi vengono rappresentati all'interno dei simboli < e >. |
| 9.999EEEE | La rappresentazione avviene in notazione scientifica (devono esserci esattamente 4 E). |
| 999V99 | Moltiplica il valore per 10^n dove n è il numero di cifre a destra di V. Ad esempio, 999V99 trasforma 1234 in 123400. |

NUMERO DI VERSIONE

Il numero di versione è il primo numero dell'identificativo di ORACLE. Se l'identificativo di ORACLE è V8.0.3.0, il numero di versione è 8.

NUMERO SEQUENZIALE DI RIGA

È un numero assegnato a una riga al suo inserimento all'interno del blocco dati di una tabella. Questo numero viene archiviato assieme alla riga ed è una parte costitutiva del ROWID.

NUMWIDTH (SQL*PLUS)

Vedere SET.

NVL

TIPO Funzione SQL

PRODOTTI Tutti, PL/SQL

VEDERE ANCHE FUNZIONI DI GRUPPO, NULL, ALTRE FUNZIONI, Capitolo 7

SINTASSI `NVL(valore, sostituto)`

DESCRIZIONE Se il valore è NULL questa funzione restituisce *sostituto*. Se il valore è diverso da NULL questa funzione restituisce *valore*. Il valore fornito può appartenere a qualsiasi tipo di dati di ORACLE. *sostituto* può essere un letterale, un'altra colonna o un'espressione (ma deve comunque essere dello stesso tipo di *valore*).

OGGETTO

Un oggetto è un elemento di un database ORACLE a cui sia possibile assegnare un nome. Gli oggetti possono dunque essere database, tabelle, indici, sinonimi, procedure e trigger.

OID

Un OID è un identificatore di un oggetto assegnato da ORACLE a tutti gli oggetti di un database. Ad esempio, ogni oggetto riga di una tabella oggetto ha un proprio OID che viene utilizzato internamente per risolvere i riferimenti agli oggetti contenuti al loro interno. ORACLE non riutilizza gli OID neanche dopo la cancellazione degli oggetti relativi. *Vedere il Capitolo 31.*

OPEN

TIPO Istruzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE CLOSE, DECLARE CURSOR, FETCH, CICLI, Capitolo 22

SINTASSI OPEN *cursor* [(*parametro*[,*parametro*]...)]

DESCRIZIONE OPEN funziona assieme a DECLARE CURSOR e FETCH. DECLARE CURSOR seleziona un'istruzione SELECT da eseguire e stabilisce un elenco di parametri (variabili PL/SQL) da utilizzare nella sua clausola where, ma non esegue la query.

È OPEN cursor che effettivamente esegue la query e mantiene i risultati in un'area temporanea da dove possono essere richiamati, una riga alla volta, con FETCH; i loro valori di colonna vengono quindi immessi all'interno di variabili locali dalla clausola INTO di FETCH. Se l'istruzione di selezione del cursore ha utilizzato dei parametri, i loro valori vengono passati all'istruzione SELECT dell'elenco di parametri di OPEN. Il numero, la posizione e il tipo di dati deve coincidere perfettamente con quelli in arrivo.

Esiste anche un metodo alternativo per associare i valori di OPEN con quelli dell'elenco di SELECT:

```
DECLARE
    cursor talbot(Moniker, Eta) is select ...
BEGIN
    open talbot(nuovo_impiegato => Moniker, 50 => Eta);
```

In questo modo nuovo_impiegato, una variabile PL/SQL caricata da una videata di introduzione dati, viene fatta puntare a Moniker e perciò lo carica qualsiasi sia il valore contenuto attualmente nella variabile. In Eta viene caricato il valore 50 e questi diventano i parametri del cursore talbot (*vedere* DECLARE CURSOR per ulteriori dettagli relativi alla parametrizzazione dei cursori).

È altresì 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 si possa chiuderlo e riaprirlo. Non si può utilizzare un cursore per cui vi sua un'istruzione OPEN pendente in un ciclo FOR.

OPEN CURSOR (interno SQL)

TIPO Comando SQL interno

PRODOTTI Precompilatori

VEDERE ANCHE CLOSE, DECLARE CURSOR, FETCH, PREPARE

SINTASSI EXEC SQL OPEN *cursor*

```
[USING {:variabile[ INDICATOR ]:variabile_indicatore
      [,:variabile[ INDICATOR ]:variabile_indicatore]... |
      DESCRIPTOR descrittore}]
```

DESCRIZIONE *cursor* è il nome di un cursore precedentemente dichiarato all'interno di un'istruzione DECLARE CURSOR. Il comando USING opzionale si riferisce all'elenco di variabili che devono essere sostituite nell'istruzione DECLARE CURSOR sulla base della posizione (il numero e il tipo delle variabili deve essere lo stesso) oppure a un descrittore risultato di un DESCRIBE precedente.

OPEN cursor alloca un cursore, definisce il campo di righe attive (le variabili host vengono sostituite all'atto dell'apertura del cursore) e posiziona il cursore appena prima della riga iniziale. Fino all'esecuzione di un FETCH non viene caricata alcuna riga. Le variabili host una volta sostituite non possono essere modificate. Per effettuare dei cambiamenti è necessario riaprire il cursore (in questo caso non è necessario che sia già stato chiuso).

ESEMPIO Questo esempio dichiara un cursore, lo apre, ne preleva delle righe e, quando queste sono terminate, chiude il cursore:

```
EXEC SQL at TALBOT declare FRED cursor
  for select DataAzione, Persona, Articolo, Importo
    from REGISTRO
    where Articolo = :Articolo
      and DataAzione = :DataAzione
      for update of Importo;

EXEC SQL open FRED;

registro: loop

  EXEC SQL fetch FRED into :Check_Date, :Buyer, :Choice, :Value;

  if sqlca.sqlcode = oracle.error.not_found
    then exit loop registro;
  end if;

end loop registro;

EXEC SQL close FRED;
```

OPERATORE

Un operatore è un carattere o una parola chiave riservata impiegata all'interno di un'espressione per effettuare un'operazione, come una somma o una comparazione, sugli elementi dell'espressione stessa. Alcuni esempi di operatori sono * (moltiplicazione), > (maggiore di) o ANY (confronta un valore con tutti quelli restituiti da una sottoquery).

OPS\$ LOGIN

Gli OPS\$ LOGIN sono particolari nomi utente di ORACLE. Alla stringa OPS\$ viene fatto precedere l'ID utente del sistema operativo in uso in modo da semplificare l'accesso a ORACLE da parte degli utenti del sistema. *Vedere l'ORACLE Server Administrator's Guide.*

OPERATORE RELAZIONALE

Un operatore relazionale è un simbolo utilizzato in un'espressione di ricerca che indichi una comparazione tra due valori. Tipico esempio è il segno di uguale in “where Importo = .10” che fa sì che vengano restituite soltanto le righe per cui l'espressione risulti vera.

OPERATORI DI QUERY

Di seguito è riportato un elenco alfabetico degli operatori di query del SQL di ORACLE. Ciascun operatore viene riportato altrove in questo glossario assieme alla propria sintassi e ad esempi di utilizzo. *Vedere anche il Capitolo 11.*

| OPERATORE | SCOPO |
|-----------|--|
| UNION | Restituisce tutte le righe di una coppia di query (senza ripetizioni). |
| UNION ALL | Restituisce tutte le righe di una coppia di query (con eventuali ripetizioni). |
| INTERSECT | Restituisce tutte le righe contemporaneamente presenti tra i risultati di due query (senza ripetizioni). |
| MINUS | Restituisce tutte le righe della prima query che non sono presenti nella seconda. |

OPERATORI DI RICERCA TESTUALE

Gli operatori seguenti vengono utilizzati all'interno della funzione CONTAINS durante le ricerche ConText. *Vedere il Capitolo 29 per esempi d'uso.*

| 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 ad AND. |
| ACCUMUL | Restituisce un record se la somma dei punteggi dei termini di ricerca supera la soglia. |
| , | Identico ad ACCUMUL. |
| 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 dev'essere messo in relazione alla vicinanza reciproca dei termini di ricerca. |

| OPERATORE | DESCRIZIONE |
|-----------|--|
| : | Identico a NEAR. |
| {} | Racchiudono parole riservate, come AND, se fanno parte di un termine di ricerca. |
| % | Carattere jolly. Sostituisce caratteri multipli. |
| - | Carattere jolly. Sostituisce un unico carattere. |
| \$ | Esegue l'espansione stem del termine da cercare prima di effettuare la ricerca. |
| ? | Effettua una corrispondenza fuzzy (che trova anche gli errori di digitazione) del termine prima di effettuare la ricerca vera e propria. |
| ! | Esegue una ricerca SOUNDEX (fonetica anglosassone). |
| 0 | Specifica l'ordine di valutazione dei criteri di ricerca. |

OPERATORI INSIEMISTICI

Gli operatori insiemistici sono UNION, INTERSECT e MINUS. Vedere anche OPERATORI DI QUERY.

OPERATORI LOGICI

TIPO Operatori SQL

PRODOTTI Tutti

VEDERE ANCHE PRECEDENZE

SINTASSI L'elenco seguente mostra tutti gli operatori logici correntemente supportati dall'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 sia con colonne che 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 relazionale come prefisso (Ad 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

| | |
|------------------|--|
| () | Forza particolari regole di precedenza nella valutazione delle espressioni o racchiude una sottoquery. |
| NOT | Completa l'espressione logica |
| AND | Combina espressioni logiche |
| OR | Combina espressioni logiche |
| UNION | Combina i risultati delle query |
| INTERSECT | Combina i risultati delle query |
| MINUS | Combina i risultati delle query |

OPERATORI SINTATTICI

TIPO Operatori SQL

PRODOTTI Tutti

VEDERE ANCHE OPERATORI LOGICI, PRECEDENZE

DESCRIZIONE Gli operatori sintattici hanno la precedenza più alta e possono comparire ovunque all'interno di istruzioni SQL. Sono elencati di seguito in ordine decrescente di precedenza. Gli operatori con la stessa precedenza vengono valutati da sinistra a destra. La maggior parte di questi operatori è descritta separatamente in questo stesso capitolo.

| OPERATORE | FUNZIONE |
|-----------|--|
| - | Continuazione di un comando SQL*PLUS. Continua un comando sulla riga seguente. |
| & | Prefisso dei parametri di un file di partenza SQL*PLUS. Al posto di &1, &2, ecc. vengono sostituiti i parametri della riga di comando. Vedere START. |
| & && | Prefisso di una variabile di sostituzione di un comando SQL in SQL*PLUS. SQL*PLUS chiederà un valore se viene riscontrata una variabile & o && indefinita. && ha l'effetto di definire la variabile e non salva il valore; '&' invece non lo fa. Vedere & e &&, DEFINE e ACCEPT. |
| : | Prefisso delle variabili di SQL*FORMS e delle variabili 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 sottoquery ed elenchi di colonne e amministra le precedenze. |
| ' | Racchiudono letterali (come stringhe di caratteri o date). Per includere un apice in una stringa se ne usino due (non si usino gli apici doppi). |

| | |
|---|--|
| " | Racchiudono una tabella o il nome dell'alias di una colonna che eventualmente contenga caratteri speciali o spazi. |
| " | Racchiudono il testo letterale di un modello di formato di TO_CHAR. |
| @ | Precede il nome di un database nel comando COPY o il nome di un link nella clausola from. |

OPZIONE DI CARICAMENTO A PERCORSO DIRETTO

SQL*LOADER durante l'inserimento dei record genera un cospicuo numero di istruzioni insert. Per evitare il sovraccarico associato a un gran numero di inserimenti è possibile utilizzare l'opzione di percorso diretto di SQL*LOADER che crea blocchi di dati preformattati e li inserisce all'interno delle tabelle. Il risultato è un drastico incremento delle prestazioni. Per utilizzare l'opzione di percorso diretto non devono essere attivate funzioni riguardanti i dati letti dal file di input.

Gli indici della tabella caricata vengono disposti in uno stato temporaneo DIRECT LOAD. Una volta completato il caricamento i vecchi valori degli indici vengono mischiati ai nuovi. Quando l'indice torna a essere valido il suo stato diventa VALID. Per minimizzare la quantità di spazio necessaria per i file temporanei conviene preordinare i dati sulla base delle colonne indicizzate. Il nome dell'indice su cui i dati sono stati preordinati dovrebbe essere specificato attraverso la clausola SORTED INDEXES del file di controllo.

Per utilizzare l'opzione di percorso diretto si specifica:

DIRECT=TRUE

come parola chiave sulla riga di comando di SQLLOAD.

OR

TIPO Operatore SQL, operatore di ricerca testuale

PRODOTTI Tutti

VEDERE ANCHE AND, OPERATORI DI RICERCA TESTUALE, Capitoli 3 e 29

DESCRIZIONE OR combina espressioni logiche in modo che il risultato sia vero se almeno una delle espressioni di partenza era vera.

ESEMPIO L'esempio seguente stampa i dati relativi sia a KEENE sia a SAN FRANCISCO:

```
select * from COMFORT
where Citta = 'KEENE' OR Citta = 'SAN FRANCISCO'
```

All'interno di ConText, l'operatore OR può essere utilizzato per ricerche relative a diversi termini. Se almeno uno dei termini viene trovato e il suo punteggio di ricerca supera il limite specificato il testo viene ritornato nella sua interezza. Vedere il Capitolo 29.

ORACLE PROGRAM INTERFACE (OPI)

L'OPI è quella parte del programma di interfaccia di ORACLE che gestisce le regole di sicurezza nello scambio di dati tra programmi utente e ORACLE stesso. Vedere USER PROGRAM INTERFACE (UPI).

ORACLE WEB APPLICATION SERVER

L'ORACLE Web Application Server (WAS) permette agli sviluppatori di accedere a database ORACLE attraverso applicazioni Web. È possibile richiamare procedure database che recuperino o manipolino i dati e ritornino i risultati agli sviluppatori.

Per gestire la formattazione adeguata dei dati onde visualizzarli attraverso un browser, WAS fornisce due package, HTF e HTP, che permettono di incorporare comandi HTML (HyperText Markup Language) all'output delle proprie procedure. I comandi HTML comunicano al browser le modalità di visualizzazione dei dati.

Una completa discussione dell'HTML esula gli scopi di questo libro. Per una trattazione completa dei package HTF e HTP si rimanda ai paragrafi omonimi in questo stesso capitolo. Nelle versioni precedenti di ORACLE, WAS era chiamato ORACLE Web Server.

ORACLE WEB SERVER

Vedere ORACLE WEB APPLICATION SERVER

ORDBMS

Vedere RELAZIONALE A OGGETTI, SISTEMA DI GESTIONE DI DATABASE

ORDER BY

TIPO Clausola SQL

PRODOTTI Tutti

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 può essere effettuato attraverso le *espressioni* (che possono essere semplici nomi di colonna o complessi insiemi di funzioni), attraverso degli *alias* (*vedere* la nota seguente) o mediante una *posizione* all'interno della clausola select (*vedere* la nota seguente).

Le righe vengono ordinate sulla prima espressione o posizione, poi sulla seconda e così via. Se non viene specificato alcuna clausola ORDER BY l'ordine di apparizione dei risultati è indeterminato e può cambiare da una query all'altra.

NOTA L'uso di posizioni di colonna nelle clausole ORDER BY dovrebbe essere abbandonato; questa caratteristica non fa più parte dello standard SQL. ORACLE continua a supportarlo per compatibilità all'indietro ma non esiste alcuna garanzia supporto nelle future versioni. Al suo posto si utilizzino gli alias di colonna.

ASC e DESC specificano se l'ordinamento debba essere ascendente o discendente e possono seguire qualsiasi espressione, posizione o alias. I valori NULL precedono gli altri negli ordinamenti ascendenti e seguono negli ordinamenti discendenti.

ORDER BY segue tutte le altre clausole a eccezione di FOR UPDATE OF.

Se ORDER BY e DISTINCT vengono specificati assieme, la clausola ORDER BY può riferirsi solo alle colonne e alle espressioni della clausola SELECT.

Quando vengono utilizzati gli operatori UNION, INTERSECT o MINUS, ORDER BY deve necessariamente utilizzare le posizioni di colonna, entrando quindi in conflitto con le modifiche allo standard SQL della nota precedente.

In un ORDER BY possono apparire solo tipi CHAR, VARCHAR2, NUMBER, DATE e il tipo speciale RowID.

ESEMPIO select Nome, Eta from LAVORATORE
order by Eta;

OTTIMIZZATORE

Un 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 36.

OUTER JOIN

Vedere JOIN per una trattazione dettagliata.

PACKAGE

Un package è un oggetto PL/SQL che raggruppa tipi di dati PL/SQL, variabili, cursori SQL, eccezioni, procedure e funzioni. Ciascun package ha una parte dichiarativa e un corpo; la prima specifica gli oggetti che possono essere manipolati durante l'uso del package. Il corpo definisce tutti gli oggetti e può contenere oggetti addizionali utilizzati solo per compiti interni. Si può modificare il corpo (ad esempio aggiungendo nuove procedure) 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 24.

PAGINA

Una pagina è una unità di archiviazione di un sistema di memorizzazione su disco. *Vedere* BLOCCO.

PAGESIZE (SQL*PLUS)

Vedere SET.

PARAMETRI

TIPO Argomento SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE &, &&, ACCEPT, DEFINE, Capitolo 15

DESCRIZIONE I parametri permettono l'esecuzione dei file di avvio passando loro dei valori dalla riga di comando. Essi vengono posti semplicemente dopo il nome del file di inizio separati gli uni gli altri da uno spazio. All'interno dei file essi sono accessibili sulla base dell'ordine in cui apparivano sulla riga di comando. &1 è il primo, &2 il secondo e così via. Le regole d'uso sono le stesse delle variabili caricate attraverso DEFINE o ACCEPT e possono essere utilizzate allo stesso modo.

Esiste comunque una limitazione: non c'è alcun modo di passare un argomento composto di più parole a una variabile singola. Ciascuna variabile può contenere una sola parola, data o numero. Se però si racchiudono i parametri tra apici le parole vengono concatenate senza particolari problemi.

ESEMPIO Si supponga di avere un file di avvio di nome fred.sql che contenga il codice SQL seguente:

```
select Nome, Eta
  from LAVORATORE
 where Eta > &1;
```

Se lo si esegue attraverso il comando seguente:

```
start fred.sql 21
```

esso produce un report su tutti i lavoratori che hanno più di 21 anni.

PARAMETRI VARIABILI

I parametri variabili sono quei parametri che possono essere impostati nel file di parametri init.ora e che influiscono sulla dimensione della System Global Area (SGA). Se il valore di questi parametri viene aumentato, la quantità di spazio richiesta dalla SGA cresce conseguentemente.

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. *Vedere PARAMETRI* per un esempio.

PARENTESI

Vedere OPERATORI LOGICI e PAROLE CHIAVE E SIMBOLI PL/SQL.

PAROLE CHIAVE E SIMBOLI PL/SQL

Questi simboli e parole chiave, elencati in altri punti di questo capitolo, sono direttamente o indirettamente collegati al PL/SQL.

| SIMBOLO | DESCRIZIONE |
|------------|---|
| () | Parentesi, utilizzate per controllare le precedenze. |
| + - * / ** | Somma, sottrazione (o negazione), moltiplicazione, divisione ed elevamento a potenza. |
| -- | Singola riga di commento. <i>Vedere COMMENT e --.</i> |
| /* */ | Commento multiriga. <i>Vedere COMMENT e /*.</i> |
| : | Terminatore. <i>Vedere TERMINATORE SQL.</i> |
| % | Prefisso degli attributi PL/SQL. |
| , | Virgola, separa elenchi di oggetti. |
| . | Punto. Separa porzioni di identificatori composti. |
| .. | Operatore di intervallo. <i>Vedere CICLO.</i> |

| SIMBOLO | DESCRIZIONE |
|-------------|---|
| @ | "chiocciola" utilizzata per riferimenti a database remoti. |
| ' | Apice singolo. Delimita una stringa di caratteri. |
| " | Doppio apice. Delimita una stringa di caratteri in modo che sia un identificativo. Il suo uso è sconsigliato. |
| [] | Utilizzato internamente da ORACLE o per racchiudere le opzioni nella descrizione sintattica dei comandi. |
| : | Due punti. Precede una variabile host. |
| <> !=~~~ ^= | Tutti significano diverso da nella logica SQL e PL/SQL. |
| < <= > >= | Minore di, minore o uguale a, maggiore di, maggiore o uguale a. Utilizzati nella logica SQL e PL/SQL. |
| => | Operatore di associazione. |
| := | Assegnazione. Assegna un valore a una variabile host o PL/SQL. Viene utilizzata nei cicli FOR. |
| | Concatenazione |
| << >> | Delimitatore di etichette di LOOP, GOTO e BLOCK. |

PAROLE CHIAVE

| | |
|---------------------|--------------------|
| %FOUND | CREATE TRIGGER |
| %ISOPEN | CURSOR |
| %NOTFOUND | DECLARE CURSOR |
| %ROWCOUNT | DECLARE EXCEPTION |
| %ROWTYPE | DECLARE VARIABLE |
| %TYPE | DELETE |
| ALTER FUNCTION | END |
| ALTER PACKAGE | EXCEPTION |
| ALTER TRIGGER | EXCEPTION_INIT |
| BEGIN | EXIT |
| BLOCK STRUCTURE | FETCH |
| CLOSE | FOR |
| COMMIT | GOTO |
| CREATE FUNCTION | IF |
| CREATE PACKAGE | INDICATOR VARIABLE |
| CREATE PACKAGE BODY | INSERT |
| CREATE PROCEDURE | LABEL |
| LOCK TABLE | SET TRANSACTION |
| LOOP | SQL CURSOR |

PAROLE CHIAVE

| | |
|-----------------|----------------------|
| NULL | SQLCODE |
| OPEN | SQLERRM |
| PACKAGE | SQLTERMINATOR |
| PRAGMA | TABLE |
| PROCEDURE | TERMINATOR |
| RAISE | TRIGGER |
| RECORD | UPDATE |
| ROLLBACK | VARIABLE DECLARATION |
| SAVEPOINT | WHILE |
| SELECT ... INTO | |

Inoltre le funzioni seguenti, identiche alle funzioni SQL omonime, possono essere utilizzate direttamente all'interno di istruzioni PL/SQL evitando quindi una chiamata SQL al database. Per apprendere le modalità operative di ciascuna di queste funzioni occorre cercarle all'interno di questo stesso capitolo:

| | |
|-------------|-------------|
| ABS | SIN |
| ACOS | SINH |
| ADD_MONTHS | SOUNDEX |
| ASIN | GREATEST_LB |
| ASCII | HEXTORAW |
| ATAN | INITCAP |
| ATAN2 | INSTR |
| CEIL | INSTRB |
| CHARTOROWID | LAST_DAY |
| CHR | LEAST |
| CONCAT | LEAST_UB |
| CONVERT | LENGTH |
| COS | LENGTHB |
| COSH | LN |
| DECODE | LOG |
| DUMP | LOWER |
| EXP | LPAD |
| FLOOR | LTRIM |
| GREATEST | SQRT |

| | |
|----------------|-------------------|
| MOD | SUBSTR |
| MONTHS_BETWEEN | SUBSTRB |
| NEW_TIME | SYSDATE |
| NEXT_DAY | TAN |
| NLS_INITCAP | TANH |
| NLS_LOWER | TO_BINARY_INTEGER |
| NLS_UPPER | TO_CHAR |
| NLSSORT | TO_DATE |
| NVL | TO_LABEL |
| POWER | TO_MULTI_BYTE |
| RAWTOHEX | TO_NUMBER |
| REPLACE | TO_SINGLE_BYTE |
| ROUND | TRANSLATE |
| ROWIDTOCHAR | TRUNC |
| RPAD | UPPER |
| RTRIM | USERENV |
| SIGN | VSIZE |

PAROLE RISERVATE

TIPO Restrizione di nomenclatura degli oggetti

PRODOTTI SQL e PL/SQL

VEDERE ANCHE NOMI DEGLI OGGETTI

DESCRIZIONE Un nome riservato ha un significato particolare per SQL e per questo motivo non può essere utilizzato come nome di un oggetto. Alcune parole chiave sono riservate in alcuni prodotti ma non in altri. L'elenco seguente cerca di visualizzare quali prodotti riservano quali parole. Nella tabella seguente, le parole chiave con gli asterischi sono riservate sia in SQL che in PL/SQL. Le parole chiave senza asterisco sono riservate solo in PL/SQL.

| | | | |
|---------|-----------|----------|-----------|
| ABORT | DELETE* | MISLABEL | ROWNUM |
| ACCEPT | DESC* | MOD | ROWS* |
| ACCESS* | DIGITS | MODE* | ROWTYPE |
| ADD* | DISPOSE | NATURAL | RUN |
| ALL* | DISTINCT* | NATURALN | SAVEPOINT |
| ALTER* | DO | NEW | SCHEMA |
| AND* | DROP* | NEXTVAL | SELECT* |
| ANY* | ELSE* | NOAUDIT* | SEPARATE |

| | | | |
|----------------|----------------|-------------|-------------|
| ARRAY | ELSIF | NOCOMPRESS* | SESSION* |
| ARRAYLEN | END | NOLOGGING* | SET* |
| AS* | ENTRY | NOT* | SHARE* |
| ASC* | EXCEPTION | NOWAIT* | SMALLINT* |
| ASSERT | EXCEPTION_INIT | NULL* | SPACE |
| ASSIGN | EXCLUSIVE* | NUMBER* | SQL |
| AT | EXISTS* | NUMBER_BASE | SQLCODE |
| AUDIT* | EXIT | OF* | SQLERRM |
| AUTHORIZATION | FALSE | OFFLINE* | START* |
| AVG | FETCH | ON* | STATEMENT |
| BASE_TABLE | FILE* | ONLINE* | STDDEV |
| BEGIN | FLOAT* | OPEN | SUBTYPE |
| BETWEEN* | FOR* | OPTION* | SUCCESSFUL* |
| BINARY_INTEGER | FORM | OR* | SUM |
| BODY | FROM* | ORDER* | SYNONYM* |
| BOOLEAN | FUNCTION | OTHERS | SYSDATE* |
| BY* | GENERIC | OUT | TABAUTH |
| CASE | GOTO | PACKAGE | TABLE* |
| CHAR* | GRANT* | PARTITION | TABLES |
| CHAR_BASE | GROUP* | PCTFREE* | TASK |
| CHECK* | HAVING* | PCTUSED* | TERMINATE |
| CLOSE | IDENTIFIED* | PLS_INTEGER | THEN* |
| CLUSTER* | IF | POSITIVE | TO* |
| CLUSTERS | IMMEDIATE* | POSITIVEN | TRIGGER* |
| COLAUTH | IN* | PRAGMA | TRUE |
| COLUMN* | INCREMENT* | PRIOR* | TYPE |
| COMMENT* | INDEX* | PRIVATE | UID* |
| COMMIT | INDEXES | PRIVILEGES* | UNION* |
| COMPRESS* | INDICATOR | PROCEDURE | UNIQUE* |
| CONNECT* | INITIAL* | PUBLIC* | UPDATE* |
| CONSTANT | INSERT* | RAISE | USE |
| CRASH | INTEGER* | RANGE | USER* |
| CREATE* | INTERFACE | RAW* | VALIDATE* |
| CURRENT* | INTERSECT* | REAL | VALUES* |
| CURRVAL | INTO* | RECORD | VARCHAR* |

| | | | |
|-----------|-------------|-----------|-----------|
| CURSOR | IS* | REF | VARCHAR2* |
| DATABASE | LEVEL* | RELEASE | VARIANCE |
| DATA_BASE | LIKE* | REMR | VIEW* |
| DATE* | LIMITED | RENAME* | VIEWS |
| DBA | LOCK* | RESOURCE* | WHEN |
| DEBUGOFF | LONG* | RETURN | WHENEVER* |
| DEBUGON | LOOP | REVERSE | WHERE* |
| DECLARE | MAX | REVOKE* | WHILE |
| DECIMAL* | MAXEXTENTS* | ROLLBACK | WITH* |
| DEFAULT* | MIN | ROW* | WORK |
| DEFINTION | MINEXTENTS* | ROWID* | WRITE |
| DELAY | MINUS | ROWLABEL* | XOR |

PARSING

Il parsing è la mappatura di un'istruzione SQL a un cursore. A questo stadio vengono effettuati diversi controlli di validazione quali le verifiche di esistenza di tutti gli oggetti richiamati, i controlli sintattici e semantici e così via. Vengono altresì prese le decisioni riguardanti esecuzione ed ottimizzazione del codice come la scelta degli indici e così via.

PARTIZIONE

Partizionare una tabella significa dividere sistematicamente le sue righe tra tabelle diverse, ciascuna con la stessa struttura. Come in ORACLE8 è possibile imporre al database di partizionare automaticamente una tabella e i relativi indici. *Vedere il Capitolo 17.*

PASSWORD

Una password è una stringa di caratteri che è necessario inserire all'atto della connessione a un database di ORACLE. Le password dovrebbero essere mantenute segrete.

PASSWORD (comando SQL*PLUS)

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE ALTER USER, Capitolo 18

DESCRIZIONE Il comando password di SQLPLUS modifica la propria password personale. Durante l'esecuzione di questo comando la password inserita dall'utente non compare mai a video. Gli utenti con diritti DBA possono modificare la password di un utente mediante lo stesso comando.

Quando si impartisce il comando password il sistema chiede di inserire prima la vecchia password e poi due volte la nuova.

ESEMPIO password
Changing password for dora
Old password:
New password:
Retype new password:

Una volta che la password sia stata cambiata con successo il sistema avvisa con il messaggio:

Password changed

PAUSE (forma 1, SQL*PLUS)

Vedere SET.

PAUSE (forma 2, SQL*PLUS)

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE ACCEPT, PROMPT, Capitolo 5

SINTASSI PAU[SE] [testo];

DESCRIZIONE PAUSE è simile a PROMPT, ma visualizza prima una riga vuota, poi una riga contenente il *testo*, quindi attende che l'utente prema INVIO. Se non si è digitato alcun *testo* PAUSE visualizza due righe vuote e attende la pressione di INVIO.

NOTA PAUSE attende la pressione di INVIO dal terminale anche quando la sorgente dei comandi sia stata dirottata da un file. Un file di avvio contenente SET TERMOOUT OFF può arrestarsi, attendere la pressione di INVIO senza fornire alcun messaggio all'utente in attesa). Si usi questo comando con molta cautela.

ESEMPIO prompt Report completo.

pause Premere INVIO per continuare.

PCTFREE

È una porzione di un blocco dati riservata ad aggiornamenti successivi sulle righe del blocco stesso.

PCTUSED

PCTUSED è la percentuale di spazio di un blocco dati che ORACLE cerca di riempire prima di allocare un nuovo blocco.

PERSONALITÀ

I server ConText sono configurati in modo da supportare diverse categorie di comandi. La raccolta di categorie supportate da un server ConText (vedere CATEGORIA) definisce la *personalità* del server. Vedere il Capitolo 30.

POLICY

Prima di creare un indice testuale di una tabella (per effettuare ricerche con ConText) è necessario definire le regole da utilizzare per la sua gestione (come ad esempio se debbano essere effettuate delle ricerche SOUNDEX). Vedere il Capitolo 30 per dettagli relativi alla creazione e alla modifica di regole per indici testuali.

POOL SQL CONDIVISO

Un'area della SGA che contiene sia la cache dei dizionari che un'area condivisa per le versioni precompilate di tutti i comandi SQL del database. È anche detto area SQL condivisa; la sua dimensione viene determinata dal parametro SHARED_POOL_SIZE del file init.ora.

POWER

TIPO Comando SQL

PRODOTTI Tutti, PL/SQL

VEDERE ANCHE SQRT, Capitolo 7

SINTASSI POWER(*valore,esponente*)

DESCRIZIONE POWER restituisce *valore* elevato all'*esponente*.

ESEMPIO POWER(2,4) = 16

PRAGMA

Una pragma-istruzione è una direttiva per il compilatore. Nonostante gli somigli e appaia all'interno dei listati dei programmi, non si tratta di uno spezzone di codice eseguibile e per questo motivo non appare all'interno del codice oggetto del blocco. Il suo compito è quello di fornire istruzioni al compilatore. Vedere EXCEPTION_INIT e il Capitolo 25.

PRECEDENZE

TIPO Regole vincolanti l'ordine di esecuzione delle operazioni

PRODOTTI Tutti

VEDERE ANCHE OPERATORI LOGICI, OPERATORI DI QUERY, Capitolo 11

DESCRIZIONE Gli operatori seguenti vengono elencati in ordine decrescente di precedenza. Gli operatori con uguale precedenza sono posti sulla stessa riga. Essi vengono inoltre valutati da sinistra a destra. Tutti gli AND sono valutati prima di qualsiasi OR.

Questi operatori vengono elencati e descritti altrove in questo stesso capitolo.

| OPERATORE | FUNZIONE |
|-----------|---|
| - | Continuazione di un comando SQL*PLUS. Continua un comando sulla riga seguente. |
| & | Prefisso dei parametri di un file di avvio di SQL*PLUS. Vedere START. |
| & && | Prefisso di sostituzione di un comando SQL in SQL*PLUS. Quando viene riscontrata una variabile & o && indefinita SQL*PLUS chiederà un valore all'utente. && definisce la variabile e ne salva il valore; '&' invece no. Vedere & e &&, DEFINE e ACCEPT. |
| : | Prefisso delle variabili host di PL/SQL. |
| . | Separatore di variabili, utilizzato da 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 sottoquery o gli elenchi di colonne. |
| ' | Racchiudono un letterale, come una stringa di caratteri o una data. Per includere un apice in una stringa se ne devono utilizzare due. |

| OPERATORE | FUNZIONE |
|-----------|---|
| " | Racchiudono l'alias di una tabella o di una colonna nel caso contenga caratteri speciali o spazi. |
| " | Racchiudono le date nella clausola TO_CHAR. |
| @ | Precede il nome di un database (in COPY) o il nome di un collegamento (nella clausola from). |
| () | Sovrascrivono la normale precedenza degli operatori. |
| + - | Segno (positivo o negativo) di un numero o di un'espressione numerica. |
| * / | Moltiplicazione e divisione. |
| + - | Somma e sottrazione. |
| | Concatenazione di caratteri. |
| NOT | Complementa il risultato logico di una espressione. |
| AND | È vera se entrambe le condizioni sono vere. |
| OR | È vera se almeno una condizione è vera. |
| UNION | Restituisce tutte le righe risultanti da due query. |
| INTERSECT | Restituisce tutte le righe identiche tra i risultati di due query. |
| MINUS | Restituisce tutte le righe della prima query che non abbiano corrispondenza nella seconda. |

PRECOMPILATORE

Il precompilatore è un programma che legge un codice sorgente particolare e riscrive un file sorgente modificato (precompilato) pronto per essere letto dal compilatore vero e proprio.

PREDICATO

Un predicato è un criterio di selezione che si basi su un solo operatore relazionale (=, !=, IS, IS NOT, >, >=) e che non contenga alcun operatore logico (AND, OR o NOT).

PREPARE (interno SQL)

TIPO Comando SQL

PRODOTTI Precompilatori

VEDERE ANCHE CLOSE, DECLARE, CURSOR, FETCH, OPEN

SINTASSI EXEC SQL PREPARE *nome_istruzione* FROM {:*stringa* | *testo*}

DESCRIZIONE PREPARE effettua il parsing SQL all'interno della variabile host *stringa* o del *testo letterale*. Inoltre assegna un *nome_istruzione* come riferimento. Se il *nome_istruzione* 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* non è il vero nome della variabile host ma un segnaposto. Un'istruzione necessita di essere preparata una volta sola, poi può essere eseguita tutte le volte che si vuole.

ESEMPIO query_string : string(1..100)
get(query_string);

```
EXEC SQL prepare FRED from :query_string;
EXEC SQL execute FRED;
```

PRINT

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE BINDING, VARIABLE; VARIABLE

SINTASSI PRINT *variabile1 variabile2 ...*

DESCRIZIONE Visualizza il valore corrente della variabile specificata (creata precedentemente con il comando VARIABLE). È possibile visualizzare il valore di più variabili con un'unica istruzione print.

PRIOR

Vedere CONNECT BY.

PRIVILEGIO

Un privilegio è un permesso rilasciato da un utente ORACLE per l'esecuzione di alcune azioni. In ORACLE nessun utente può eseguire qualsiasi 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 di esecuzione di comandi di definizione e controllo dei dati quali 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 comandi CREATE, ALTER e DROP. I privilegi che contengono la parola chiave ANY implicano che l'utente può esercitare il privilegio su qualsiasi schema su cui sia stato rilasciato l'attributo e non solo sui propri.

Alcuni dei privilegi semplicemente forniscono il permesso di eseguire il comando indicato e non necessitano di ulteriori spiegazioni. Altri, invece, vengono spiegati brevemente.

| PRIVILEGIO | PERMESSO RELATIVO |
|---------------------|--|
| ALTER ANY CLUSTER | ALTER CLUSTER |
| ALTER ANY INDEX | ALTER INDEX |
| ALTER ANY PROCEDURE | ALTER PROCEDURE, ALTER FUNCTION, ALTER PACKAGE |
| ALTER ANY ROLE | ALTER ROLE |
| ALTER ANY SEQUENCE | ALTER SEQUENCE |
| ALTER ANY SNAPSHOT | ALTER SNAPSHOT |
| ALTER ANY TABLE | ALTER TABLE |
| ALTER ANY TRIGGER | ALTER TRIGGER |
| ALTER ANY TYPE | ALTER TYPE |
| ALTER DATABASE | ALTER DATABASE |

| PRIVILEGIO | PERMESSO RELATIVO |
|-----------------------------|---|
| ALTER PROFILE | ALTER PROFILE |
| ALTER RESOURCE COST | ALTER RESOURCE COST |
| ALTER ROLLBACK SEGMENT | ALTER ROLLBACK SEGMENT |
| ALTER SESSION | ALTER SESSION |
| ALTER SYSTEM | ALTER SYSTEM |
| ALTER TABLESPACE | ALTER TABLESPACE |
| ALTER USER | ALTER USER |
| ANALYZE ANY | ANALYZE |
| AUDIT ANY | AUDIT (FORMATO 1) |
| AUDIT SYSTEM | AUDIT (FORMATO 2) |
| BACKUP ANY TABLE | Permette di esportare degli oggetti da qualsiasi schema |
| BECOME USER | Permette di importare oggetti da qualsiasi schema |
| COMMENT ANY TABLE | COMMENT |
| CREATE ANY CLUSTER | CREATE CLUSTER |
| CREATE ANY DIRECTORY | CREATE DIRECTORY |
| CREATE ANY INDEX | CREATE INDEX |
| CREATE ANY LIBRARY | CREATE LIBRARY |
| CREATE ANY PROCEDURE | CREATE PROCEDURE |
| CREATE ANY SEQUENCE | CREATE SEQUENCE |
| CREATE ANY SNAPSHOT | CREATE SNAPSHOT |
| CREATE ANY SYNONYM | CREATE SYNONYM |
| CREATE ANY TABLE | CREATE TABLE |
| CREATE ANY TRIGGER | CREATE TRIGGER |
| CREATE ANY TYPE | CREATE TYPE, CREATE TYPE BODY |
| CREATE ANY VIEW | CREATE VIEW |
| CREATE CLUSTER | CREATE CLUSTER |
| CREATE DATABASE LINK | CREATE DATABASE LINK |
| CREATE DIRECTORY | CREATE DIRECTORY |
| CREATE LIBRARY | CREATE LIBRARY |
| CREATE PROCEDURE | CREATE PROCEDURE, CREATE FUNCTION, CREATE PACKAGE |
| CREATE PROFILE | CREATE PROFILE |
| CREATE PUBLIC DATABASE LINK | CREATE PUBLIC DATABASE LINK |
| CREATE PUBLIC SYNONYM | CREATE PUBLIC SYNONYM |
| CREATE ROLE | CREATE ROLE |

| PRIVILEGIO | PERMESSO RELATIVO |
|---------------------------|--|
| CREATE ROLLBACK SEGMENT | CREATE ROLLBACK SEGMENT |
| CREATE SESSION | CREATE SESSION (connessione al database) |
| CREATE SEQUENCE | CREATE SEQUENCE |
| CREATE SNAPSHOT | CREATE SNAPSHOT |
| CREATE SYNONYM | CREATE SYNONYM |
| CREATE TABLE | CREATE TABLE |
| CREATE TABLESPACE | CREATE TABLESPACE |
| CREATE TRIGGER | CREATE TRIGGER |
| CREATE TYPE | CREATE TYPE, CREATE TYPE BODY |
| CREATE USER | CREATE USER |
| CREATE VIEW | CREATE VIEW |
| DELETE ANY TABLE | DELETE (da viste e tabelle) |
| DROP ANY CLUSTER | DROP CLUSTER |
| DROP ANY DIRECTORY | DROP DIRECTORY |
| DROP ANY INDEX | DROP INDEX |
| DROP ANY LIBRARY | DROP LIBRARY |
| DROP ANY PROCEDURE | DROP PROCEDURE, DROP FUNCTION, DROP PACKAGE |
| DROP ANY ROLE | DROP ROLE |
| DROP ANY SEQUENCE | DROP SEQUENCE |
| DROP ANY SNAPSHOT | DROP SNAPSHOT |
| DROP ANY SYNONYM | DROP SYNONYM |
| DROP ANY TABLE | DROP TABLE |
| DROP ANY TRIGGER | DROP TRIGGER |
| DROP ANY TYPE | DROP TYPE |
| DROP ANY VIEW | DROP VIEW |
| DROP LIBRARY | DROP LIBRARY |
| DROP PROFILE | DROP PROFILE |
| DROP PUBLIC DATABASE LINK | DROP PUBLIC DATABASE LINK |
| DROP PUBLIC SYNONYM | DROP PUBLIC SYNONYM |
| DROP ROLLBACK SEGMENT | DROP ROLLBACK SEGMENT |
| DROP TABLESPACE | DROP TABLESPACE |
| DROP USER | DROP USER |
| EXECUTE ANY PROCEDURE | Permette di eseguire qualsiasi funzione o procedura e di riferirsi a qualsiasi package di variabili pubbliche. |

| PRIVILEGIO | PERMESSO RELATIVO |
|-----------------------|---|
| EXECUTE ANY TYPE | Permette di utilizzare tutti i tipi di dati astratti e i metodi relativi. |
| FORCE ANY TRANSACTION | Permette di forzare la ricezione o di rifiutare qualsiasi transazione dubbia (vedere ACQUISIZIONE IN DUE FASI). |
| FORCE TRANSACTION | Permette di forzare o rifiutare transazioni utente dubbie. |
| GRANT ANY PRIVILEGE | Concede i privilegi di sistema. |
| GRANT ANY ROLE | Concede un ruolo. |
| INSERT ANY TABLE | INSERT |
| LOCK ANY TABLE | LOCK TABLE |
| MANAGE TABLESPACE | Permette di gestire i tablespace sia in linea sia fuori linea. |
| READUP | Permette l'esecuzione di query di dati con classi di accesso più elevate di quelle della sessione in corso (Trusted ORACLE). |
| RESTRICTED SESSION | Permette di accedere al sistema durante le sessioni ristrette di Server Manager. |
| SELECT ANY SEQUENCE | Seleziona dalle sequenze. |
| SELECT ANY TABLE | SELECT |
| UNLIMITED TABLESPACE | Permette di superare le quote assegnate. |
| UPDATE ANY TABLE | UPDATE |
| WRITEDOWN | Permette di creare, modificare, cancellare, inserire, aggiornare o tagliare gli oggetti la cui classe di accesso sia minore di quella della sessione corrente (Trusted ORACLE). |
| WRITEUP | Permette di creare, modificare, cancellare, inserire, aggiornare o tagliare gli oggetti la cui classe di accesso sia maggiore di quella della sessione corrente (Trusted ORACLE). |

I privilegi di oggetto possono essere applicati solo ad alcuni tipi di oggetti. La tabella seguente evidenza le relazioni. Il privilegio REFERENCES permette a un utente di creare una relazione che si riferisce alla tabella di base.

| PRIVILEGIO | TABELLE | VISTE | SEQUENZE | PROCEDURE FUNZIONI PACKAGE LIBRERIE TIPI | SNAPSHOT | DIRECTORY |
|------------|---------|-------|----------|--|----------|-----------|
| ALTER | X | | X | | X | |
| DELETE | X | X | | | | |
| EXECUTE | | | | X | | |
| INDEX | X | | | | | |
| INSERT | X | X | | | | |
| READ | | | | | X | |

| PRIVILEGIO | TABELLE | VISTE | SEQUENZE | PROCEDURE FUNZIONI PACKAGE LIBRERIE TIPI | SNAPSHOT | DIRECTORY |
|------------|---------|-------|----------|--|----------|-----------|
| REFERENCES | X | | | | | |
| SELECT | X | X | X | | X | |
| UPDATE | X | X | | | X | |

Come traspare dalla tabella precedente, è possibile operare selezioni su tabelle, viste, sequenze e snapshot (cioè dalla vista locale il cui nome compare dopo lo snapshot; *vedere* il Capitolo 28).

PRO*C

Il PRO*C è un'estensione del C che permette di sviluppare maschere di accesso a database ORACLE. Un precompilatore converte il codice PRO*C in C normale, pronto per essere compilato.

PRO*COBOL

Il PRO*COBOL è un'estensione del COBOL che permette di sviluppare maschere di accesso a database ORACLE. Un precompilatore converte il codice PRO*COBOL in COBOL standard, pronto per essere compilato.

PRO*FORTRAN

Il PRO*FORTRAN è un'estensione del FORTRAN che permette di sviluppare maschere di accesso a database ORACLE. Un precompilatore converte il codice PRO*FORTRAN in FORTRAN standard, pronto per essere compilato.

PROCEDURA

Una procedura è un insieme di istruzioni (spesso combinazioni di comandi SQL e PL/SQL) che, componendo un sottoprogramma di uso frequente, vengono salvate all'interno di librerie. *Vedere* CREATE PROCEDURE.

PROCESSI SERVER

I processi SERVER lavorano alle spalle dei processi utente.

PROCESSO PMON

Il Process MONitor è un processo attivo in background utilizzato per recuperare le situazioni in cui un processo si interrompa durante l'accesso al database. *Vedere* PROCESSO IN BACKGROUND.

PROCESSO SINGOLO

La modalità a processo singolo è una modalità operativa che permette l'accesso al database da parte di un solo utente.

PRODUCT_USER_PROFILE

Un PRODUCT_USER_PROFILE è una tabella di sistema di ORACLE utilizzata per limitare l'uso di taluni prodotti ORACLE o per disabilitare uno o più comandi SQL. Vedere la guida *SQL*Plus User's Guide and Reference*.

PROFILO

Un profilo è una raccolta di impostazioni ORACLE che limitano le risorse disponibili agli utenti. Vedere CREATE PROFILE e il Capitolo 18.

PROMPT

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE ACCEPT

SINTASSI PROMPT [*testo*]

DESCRIZIONE Questo comando visualizza il *testo* sullo schermo. Se non viene specificato alcun testo viene visualizzata una riga vuota. Per visualizzare una variabile vedere PRINT.

PSEUDOCOLONNE

TIPO Definizioni

PRODOTTI Tutti

DESCRIZIONE Una pseudocolonna è qualcosa che, pur producendo valori quando è selezionata, non è effettivamente una colonna. Esempi tipici sono RowID e SysDate. Quelle che seguono sono le pseudocolonne supportate attualmente da ORACLE:

| PSEUDOCOLONNA | VALORE RESTITUITO |
|------------------|---|
| sequence.CurrVal | Il valore corrente della sequenza specificata. |
| Level | 1 per un nodo radice, 2 per i nodi figli della radice ecc. In pratica riporta la profondità del nodo indicato. |
| sequence.NextVal | Il valore successivo della sequenza indicata. Incrementa la sequenza automaticamente. |
| NULL | Un valore NULL. |
| RowID | 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 ritornata una riga dalla selezione di una tabella. Il RowNum della prima riga è 1, quello della seconda è 2 e così via. Vedere RowNum in questo glossario e il Capitolo 16 per ulteriori dettagli. |
| SysDate | Data e ora correnti. |
| UID | Lo User ID. Un numero unico assegnato a ciascun utente. |
| User | L'identificativo utente dell'utente attuale. |

PUBLIC

Public ha due significati, descritti di seguito.

- Qualcosa che sia stato reso public è visibile a tutti gli utenti. Possono essere public i sinonimi e le connessioni a database remoti. 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 di utenti di un dato database, il nome di tale gruppo.

QUERY

Una query è un'istruzione SQL che preleva 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 sottoquery che viene eseguita reiteratamente, per ciascun valore di una riga candidata selezionata dalla query principale. L'esito di ogni esecuzione della sottoquery dipende dai valori di uno o più campi all'interno della riga candidata; in altre parole, la sottoquery è correlata con la query principale. *Vedere* il Capitolo 11.

QUERY DISTRIBUITA

Una query distribuita è una richiesta che seleziona dei dati a partire da database residenti su nodi diversi di una rete.

QUERY GENITORE

La query genitore è la query più esterna (quella che produce il risultato) in un costrutto che contenga una o più sottoquery. *Vedere* QUERY PRINCIPALE.

QUERY PRINCIPALE

Una query principale è la query più esterna di una query contenente altre sottoquery. È la query la cui colonna produce il risultato finale.

QUIT

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

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 allo spazio utilizzabile da ciascun utente all'interno del database. Vedere CREATE USER e ALTER USER.

RAISE

TIPO Istruzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE DECLARE EXCEPTION, EXCEPTION, EXCEPTION_INIT

SINTASSI RAISE [*eccezione*]

DESCRIZIONE RAISE scatena un'eccezione sulla base di una condizione assegnata. L'eccezione dev'essere stata dichiarata esplicitamente o appartenere alle eccezioni di sistema come DIVIDE_BY_ZERO (vedere EXCEPTION per ottenere un elenco completo).

L'istruzione RAISE sposta il controllo alla sezione EXCEPTION del blocco corrente nella quale, presumibilmente, sarà presente un controllo sul motivo dello scatenamento dell'eccezione. Se non è presente alcuna sezione EXCEPTION, il flusso di programma salta alla sezione EXCEPTION più vicina (quella del blocco immediatamente più esterno di un blocco ammodatp). Se non viene riscontrato alcun programma di gestione delle eccezioni il controllo ritorna al programma o all'ambiente che ha richiamato PL/SQL 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 forzare la gestione dell'eccezione corrente da parte del blocco EXCEPTION più esterno. Se ad esempio fosse avvenuto un errore di tipo NOT_LOGGED_ON e il blocco di eccezione locale contenesse le righe seguenti:

```
when not_logged_on  
      then raise;
```

l'esecuzione salta alla sezione EXCEPTION del blocco immediatamente più esterno. Il vantaggio consiste nel fatto che la gestione di una certa classe di errori può essere demandata a un unico blocco EXCEPTION.

RAMI

I rami rappresentano una serie di nodi interconnessi di un albero logico. Vedere CONNECT BY.

RAW, TIPO DI DATI

Una colonna RAW contiene dati binari in un qualsiasi formato compatibile con il computer host. La colonne RAW sono utili nell'archiviazione di dati binari (non corrispondenti a caratteri).

RAWTOHEX

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE HEXTORAW

SINTASSI RAWTOHEX(*stringa_binaria*)

DESCRIZIONE RAWTOHEX converte una stringa di numeri binari in una stringa di cifre esadecimali

RDBMS

Vedere RELAZIONALE, SISTEMA DI GESTIONE DI DATABASE

RECORD

Record è un sinonimo di riga.

RECORD (PL/SQL)

TIPO Tipi di dati PL/SQL

PRODOTTI Tutti

VEDERE ANCHE TABELLA (PL/SQL), TIPI DI DATI

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 le nuove variabili. I 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 (ivi compresi altri RECORD ma escludendo le tabelle), o essere un riferimento a tipo di una particolare colonna di una data tabella. Ogni campo può inoltre possedere l'attributo NOT NULL che specifica che il campo deve sempre avere un valore significativo.

Ci si può riferire ai singoli campi di un record utilizzando la classica notazione separata da punti.

ESEMPIO

```
type SkillRecord is record(name char(25), compito
                           COMPITOLAVORATORE.Compito%TYPE);
SkillRecord MySkill;
MySkill.Nome = 'DICK JONES';
MySkill.Compito = 'SMITHY';
```

RECOVER

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER DATABASE, ORACLE Server Administrator's Guide

SINTASSI RECOVER [AUTOMATIC] [FROM '*locazione*']

```
{ { [ [STANDBY] DATABASE] [ UNTIL CANCEL
                           | UNTIL TIME data
                           | UNTIL CHANGE intero
                           | USING BACKUP CONTROLFILE] ...
| TABLESPACE tablespace [, tablespace] ...
| DATABASE 'nomefile' [, 'nomefile'] ...
| LOGFILE 'nomefile'
| CONTINUE [DEFAULT]
| CANCEL} ] [PARALLEL clausola_parallel] }
```

DESCRIZIONE La clausola RECOVER del comando ALTER DATABASE recupera un database utilizzando diverse opzioni: il recupero AUTOMATIC genera automaticamente i nomi dei file di ripristino; l'opzione FROM specifica la posizione del gruppo di file di ripristino e dev'essere un percorso di directory valido.

L'opzione DATABASE ripristina completamente il database ed è l'opzione di default. L'alternativa UNTIL CANCEL recupera il database finché non si interrompe

l'operazione con l'opzione CANCEL. L'alternativa UNTIL TIME ricupera i database fino alla data specificata. L'alternativa UNTIL CHANGE ricupera un numero di cambiamenti di sistema (un numero unico assegnato a ciascuna transazione) specificato da un numero intero.

L'alternativa USING BACKUP CONTROLFILE ripristina il database applicando il redo log a un file di controllo del backup.

L'opzione TABLESPACE consente di recuperare solo i tablespace indicati. L'opzione DATAFILE recupera solo i file di dati specificati. L'opzione LOGFILE applica il file di log specificato. L'opzione CONTINUE continua ad applicare il file di redo, CANCEL termina il ripristino.

RECSEP (SQL*PLUS)

Vedere SET.

RECSEPCHAR (SQL*PLUS)

Vedere SET.

RECUPERO DEI SUPPORTI

Si tratta dell'operazione di recupero di guasti hardware che possono pregiudicare lettura, scrittura e aggiornamento dei dati 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 due programmi. *Vedere anche* RECUPERO DEI SUPPORTI.

REDO LOG

Un redo log è un rapporto sequenziale delle azioni che dovrebbero essere effettuate nuovamente al database se le prime non fossero state scritte su disco con successo. Il rapporto consiste di almeno due file. Quando uno dei due è pieno si comincia ad utilizzare l'altro. Se entrambi sono pieni se ne svuota uno per riutilizzarlo.

REDO LOG FUORI LINEA

Se si utilizza il redo log in modalità ARCHIVELOG, il redo log fuori linea è un file di rapporto.

REDO LOG IN LINEA

Il redo log in linea contiene i file di rapporto che non sono ancora stati archiviati. Possono essere aperti per tenere conto di un'attività di salvataggio ancora attiva o essere semplicemente in attesa di archiviazione.

REDO LOG SEQUENCE NUMBER

Il redo log sequence number è un numero utilizzato per identificare lo stato di avanzamento delle operazioni di ripristino.

REF

È un tipo di dati utilizzato con tabelle oggetto. Vedere il Capitolo 31.

REGOLA DI INTEGRITÀ REFERENZIALE

Una regola di integrità referenziale è un vincolo di integrità che forza l'integrità referenziale.

RELAZIONALE, SISTEMA DI GESTIONE DI DATABASE

Un RDBMS è un programma improntato all'archiviazione e al recupero di dati di qualsiasi tipo mediante l'archiviazione in tabelle composte di una o più unità di informazione (righe), ciascuna contenente lo stesso insieme di tipi di dati (colonne). ORACLE è un RDBMS.

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). Vedere il Capitolo 4 per una descrizione dell'implementazione dell'ORDBMS di ORACLE.

RELAZIONE

Vedere TABELLA e il Capitolo 1.

REMARK

TIPO Istruzione SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE /* */, --, DOCUMENTO, Capitolo 5

SINTASSI REM[ARK] *testo*

DESCRIZIONE REMARK precede un commento (di una sola riga) 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 di default delle colonne

REM per questo campo di dati:

column DataAzione format a6 trunc

RENAME

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE COPY, CREATE SYNONYM, CREATE TABLE, CREATE VIEW

SINTASSI RENAME *vecchio* TO *nuovo*

DESCRIZIONE RENAME rinomina tabelle, viste e sinonimi. Non sono necessari apici attorno ai nomi. Il nuovo nome non deve essere né una parola chiave riservata né il nome di un oggetto preesistente di proprietà dell'utente stesso.

ESEMPIO La riga seguente cambia il nome della tabella LAVORATORE in IMPIEGATO:

```
rename LAVORATORE to IMPIEGATO;
```

REPFOOTER

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE ACCEPT, BTITLE, DEFINE, PARAMETRI, REPHEADER, Capitolo 13

SINTASSI REPFOOTER [opzione [*testo|VARIABILE*]... | OFF|ON]

DESCRIZIONE REPFOOTER (REPort FOOTER) inserisce un pedice sull'ultima pagina di un rapporto. OFF e ON disabilitano o riabilitano la visualizzazione del testo senza modificarne il contenuto. REPFOOTER da solo mostra le opzioni correnti e il valore di *testo* o *variabile*.

variabile è il nome di una variabile definita dall'utente o gestita dal sistema come SQL.LNO, il numero di riga corrente; SQL.PNO, il numero di pagina corrente; SQL.RELEASE, il numero di versione di ORACLE; SQL.SQLCODE, il codice di errore corrente e SQL.USER, il nome utente.

Le opzioni possibili sono elencate di seguito.

- PAGE stampa il piede di pagina del report nell'ultima pagina.
- COL[UMN] *n* salta direttamente alla posizione *n* dal margine sinistro della riga corrente.
- S[KIP] *n* visualizza *n* righe vuote. Se non viene specificato alcun numero viene stampata una sola riga vuota. Se *n* vale 0 non viene visualizzata alcuna riga vuota e la posizione di stampa corrente diventa la prima della riga corrente.
- TAB *n* effettua *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 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.
- FORMAT specifica il modello di formato che controllerà la visualizzazione di testo e variabili e segue la stessa sintassi della clausola FORMAT del comando COLUMN (come FORMAT A12 e FORMAT \$999,990.99). Ogni volta che compare un nuovo comando FORMAT si ha la sovrascrittura del precedente. Se non viene specificato alcun modello FORMAT, viene utilizzato quello di SET NUMFORMAT. Se neanche questo è stato impostato, viene utilizzato il modello di default di SQL*PLUS.

Le date vengono stampate sulla base del formato di default a meno che non si utilizzi una variabile e la si riformatti con TO_CHAR.

REPHEADER

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE ACCEPT, BTITLE, DEFINE, PARAMETRI, REPFOOTER, Capitolo 13

SINTASSI REPH[EADER] [opzione [*testo|VARIABILE*]... | OFF|ON]

DESCRIZIONE REPHEADER (REPort HEADER) inserisce un apice sull'ultima pagina di un rapporto. OFF e ON disabilitano o riabilitano la visualizzazione del testo

senza modificarne il contenuto. REPHEADER da solo mostra le opzioni correnti e il valore di *testo* o *variabile*.

variabile è il nome di una variabile definita dall'utente o gestita dal sistema come SQL.LNO, il numero di riga corrente; SQL.PNO, il numero di pagina corrente; SQL.RELEASE, il numero di versione di ORACLE; SQL.SQLCODE, il codice di errore corrente e SQL.USER, il nome utente.

Le opzioni possibili sono elencate di seguito.

- PAGE stampa l'intestazione del report nell'ultima pagina.
- COL[UMN] *n* salta direttamente alla posizione *n* dal margine sinistro della riga corrente.
- S[KIP] *n* visualizza *n* righe vuote. Se non viene specificato alcun numero viene stampata una sola riga vuota. Se *n* vale 0 non viene visualizzata alcuna riga vuota e la posizione di stampa corrente diventa la prima della riga corrente.
- TAB *n* effettua *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 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.
- FORMAT specifica il modello di formato che controllerà la visualizzazione di testo e variabili e segue la stessa sintassi della clausola FORMAT del comando COLUMN (come FORMAT A12 e FORMAT \$999,990.99). Ogni volta che compare un nuovo comando FORMAT si ha la sovrascrittura del precedente. Se non viene specificato alcun modello FORMAT viene utilizzato quello di SET NUMFORMAT. Se neanche questo è stato impostato viene utilizzato il modello di default di SQL*PLUS.

Le date vengono stampate sulla base del formato di default a meno che non si utilizzi una variabile e la si riformatti con TO_CHAR.

REPLACE

TIPO Funzione SQL

PRODOTTI Tutti, PL/SQL

VEDERE ANCHE CARATTERI, FUNZIONI; TRANSLATE

SINTASSI REPLACE(*stringa*,*se*,*allora*)

DESCRIZIONE REPLACE sostituisce uno o più caratteri di una stringa con 0 o più caratteri. *se* è una stringa di uno o più caratteri. Ogni sua ricorrenza all'interno di *stringa* viene sostituita dal contenuto di *allora*.

ESEMPIO

```
REPLACE('ADAH','A','BLAH') = BLAHDBLAHH
REPLACE('GEORGE','GE',null) = OR
REPLACE('BOB','BO','TA') = TAB
```

RETE

Una rete è una connessione tra due o più computer in modo da permettere lo scambio diretto di informazioni.

REVOKE (forma 1, privilegi di sistema e ruoli)

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE DISABLE, ENABLE, GRANT, REVOKE (forma 2), PRIVILEGI, Capitolo 18

SINTASSI REVOKE {privilegio_di_sistema | ruolo}
 [,{privilegio_di_sistema | ruolo}]...
 FROM {utente | ruolo | PUBLIC}
 [,{utente | ruolo | PUBLIC}]...

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 rimuove un privilegio da un utente, questi perde il permesso di effettuare alcune operazioni. Se si rimuove un privilegio da un ruolo, nessuno degli utenti con quel ruolo è più in grado di eseguire le operazioni relative al privilegio.

Se si revoca un ruolo da un utente, questo non può riabilitarlo da sé (*vedere* ENABLE) ma può continuare a esercitare il proprio privilegio nella sessione corrente.

REVOKE (forma 2, privilegi degli oggetti)

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE GRANT, REVOKE (forma 1), PRIVILEGI, Capitolo 18

SINTASSI REVOKE privilegio[,privilegio]...
 ON [utente.]oggetto
 FROM {utente | ruolo | PUBLIC}
 [,{utente | ruolo | PUBLIC}]...
 [CASCADE CONSTRAINTS]

DESCRIZIONE Il comando REVOKE rimuove un privilegio relativo a un oggetto da un utente o da un ruolo. 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 siano contenuti in altri ruoli direttamente o indirettamente correlati all'utente.

La clausola CASCADE CONSTRAINTS rimuove qualsiasi regola di integrità dei riferimenti definita dallo o dagli utenti in possesso del ruolo.

RIGA

Riga ha due definizioni.

- Un insieme di campi di una tabella; ad 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.

RISORSA

Risorsa è il termine generale che denota un oggetto logico o una struttura fisica di un database. Le risorse possono essere bloccate. Quelle direttamente accessibili da parte degli utenti sono le tabelle e le righe. Le risorse bloccabili dall'RDBMS sono molteplici e comprendono anche dizionari, cache e file.

RESOURCE è il nome di un ruolo standard di ORACLE. *Vedere* RUOLO.

RUOLO

Un ruolo è un insieme di privilegi che un utente può concedere ad altri utenti. ORACLE dispone di otto ruoli di default. I primi tre sono i più usati. Di seguito sono elencati i ruoli di default assieme ai privilegi associati (*vedere PRIVILEGI*).

| RUOLO | PRIVILEGI |
|----------------------|---|
| CONNECT | ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW |
| RESOURCE | CREATE CLUSTER, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE |
| DBA | Tutti i privilegi di sistema WITH ADMIN OPTION, ruolo EXP_FULL_DATABASE, ruolo IMP_FULL_DATABASE |
| EXP_FULL_DATABASE | SELECT ANY TABLE, BACKUP ANY TABLE, INSERT, UPDATE, DELETE ON SYS.INCEXP, SYS.INCVID, SYS.INCFIL |
| IMP_FULL_DATABASE | BECOME USER |
| DELETE_CATALOG_ROLE | DELETE su SYS.AUD\$ |
| EXECUTE_CATALOG_ROLE | EXECUTE sulle viste del dizionario di dati |
| SELECT_CATALOG_ROLE | SELECT sulle viste del dizionario di dati |

Vedere il Capitolo 18 per esempi riguardanti la creazione e l'uso dei ruoli.

ROLL FORWARD

Si tratta di un meccanismo per cui vengono riapplicate le modifiche apportate al database e annullate. Talvolta serve per il recupero dei supporti o di istanza. Il redo log contiene le annotazioni redo utilizzate per il roll forward.

ROLLBACK

Il rollback elimina parte del lavoro effettuato nel corso della transazione corrente (fino all'ultimo COMMIT o SAVEPOINT).

ROLLBACK (forma 1, SQL)

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE COMMIT, SET AUTOCOMMIT, Capitolo 14

SINTASSI ROLLBACK [WORK] {[TO [SAVEPOINT] *savepoint*] | [FORCE *testo*]}

DESCRIZIONE ROLLBACK annulla tutti i cambiamenti effettuati alle tabelle del database dall'ultimo salvataggio e toglie tutti i blocchi applicati alle tabelle. Ogni volta che si interrompe una transazione, che avviene un errore di esecuzione o un problema all'alimentazione viene generato un rollback automatico. ROLLBACK non annulla solo l'ultima operazione di insert, update o delete, ma tutte quelle occorse dall'ultimo COMMIT. Questo permette di trattare separatamente vari spezzoni di lavoro e salvare solo quando tutte le modifiche sono state effettuate.

Se si specifica un SAVEPOINT, ROLLBACK salta a un punto specifico (etichettato) dell'elenco di transazioni. Cancella qualsiasi altro SAVEPOINT che incontra e annulla tutti i blocchi di riga occorsi dopo la creazione del SAVEPOINT.

Se si specifica FORCE, ROLLBACK effettua un rollback automatico di tutte le transazioni dubbie identificate tramite il testo che corrisponde a un identificativo di transazione della vista DBA_2PC_PENDING.

NOTA Se SET AUTOCOMMIT è attivo, però, qualsiasi insert, update o delete salva immediatamente tutte le eventuali modifiche pendenti sul database. Se si digita ROLLBACK si ottiene come risposta: ROLLBACK COMPLETE che non è assolutamente significativa. I cambiamenti permangono dato che ROLLBACK riporta alla situazione dell'ultimo salvataggio, di fatto avvenuto dopo l'ultimo cambiamento.

ALTER, AUDIT, CONNECT, CREATE, DISCONNECT, DROP, EXIT, GRANT, NOAUDIT, QUIT, REVOKE e SAVE provocano tutte un COMMIT immediato.

ROLLBACK (forma 2, interno SQL)

TIPO Comando SQL interno

PRODOTTI Precompilatori

VEDERE ANCHE COMMIT, SAVEPOINT, SET TRANSACTION

SINTASSI EXEC SQL [AT *database* | :*variabile*]

ROLLBACK [WORK]

{[TO [SAVEPOINT] *savepoint*] [RELEASE] |

[FORCE *testo*]}

DESCRIZIONE ROLLBACK termina la transazione corrente, annulla tutti i cambiamenti della transazione in corso e rilascia i blocchi ai file ma non modifica né le variabili host né il flusso di esecuzione del programma. *database* indica il nome del database su cui operare. Nel caso non venga indicato viene utilizzato il database di default dell'utente. SAVEPOINT permette di effettuare un rollback di tipo SAVEPOINT (*vedere* SAVEPOINT).

FORCE effettua il rollback manuale di una transazione dubbia. WORK è assolutamente opzionale e viene utilizzato solo per migliorare la leggibilità.

Sia ROLLBACK che COMMIT dispongono dell'opzione RELEASE. Questa dev'essere specificata dopo l'ultima transazione altrimenti i blocchi ai file possono interferire col lavoro degli altri utenti. Si ha un ROLLBACK automatico (comprendendo RELEASE) qualora il programma non si concluda normalmente.

RADICE

In una tabella ad albero, il nodo radice è la sommità dell'albero. È una riga che non ha genitore e i cui figli, nipoti ecc. costituiscono l'intero albero. Nelle query ad albero la radice è la riga specificata nella clausola START WITH.

ROUND (forma 1, data)

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE DATA, FUNZIONI; ROUND (NUMERO), TRUNC, Capitolo 8

SINTASSI ROUND (*data*, 'formato')

DESCRIZIONE ROUND arrotonda una data in base al formato specificato. Se non viene specificato alcun formato la data viene arrotondata alle 12 A.M. del giorno stesso, come nel caso di 12:00:00 P.M. (mezzogiorno esatto) di oggi, o del giorno successivo se è già pomeriggio. La data risultante avrà come ora le 12 A.M.

Formati disponibili per gli arrotondamenti

| FORMATO | SIGNIFICATO |
|--------------------------------|--|
| cc,scc | secolo (arrotonda alla mezzanotte del primo gennaio del secolo successivo). |
| syear,syy,y.yy,yyy,yyyy e year | anno (arrotonda alla mezzanotte del primo gennaio dell'anno successivo). |
| q | quarto (arrotonda alla mezzanotte del 16 del secondo mese del quarto d'anno, indipendentemente dal numero di giorni del mese). |
| month,mon,mm | mese (arrotonda alla mezzanotte esatta del 16 del mese, a prescindere dal numero di giorni del mese). |
| ww | arrotonda al lunedì più vicino (vedere l'elenco seguente). |
| w | arrotonda al primo giorno uguale a quello del primo del mese (vedere l'elenco seguente). |
| ddd,dd,j | arrotonda alla mezzanotte del giorno successivo. È il formato di default se non ne viene specificato altro. |
| day,dy,d | arrotonda alla domenica successiva (primo giorno della settimana). |
| hh,hh12,hh24 | arrotonda all'ora intera successiva. |
| Mi | arrotonda al minuto intero successivo. |

ww produce la data del lunedì più vicino (con l'ora impostata sulla mezzanotte). Questo significa che essendo una settimana di 7 giorni, qualsiasi data fino a 3 giorni e mezzo dopo lunedì (cioè giovedì alle 11:59:59) o quelle fino a 3 giorni e mezzo prima di lunedì (cioè il giovedì precedente da mezzogiorno in poi), viene arrotondata a lunedì a mezzanotte.

w funziona in maniera analoga tranne che per il fatto che produce la data del più vicino giorno uguale al primo del mese. Ad esempio, se il primo del mese era un venerdì qualsiasi data fino a 3 giorni e mezzo prima di venerdì (il lunedì precedente a mezzogiorno) e fino a 3 giorni e mezzo dopo (il lunedì successivo alle 23:59:59), vengono arrotondate alla mezzanotte di venerdì.

Quando ww e w effettuano degli arrotondamenti, l'ora e la data fornite vengono confrontate con una data (lunedì oppure il primo giorno del mese) a mezzanotte. Il risultato è esso stesso una data con l'ora impostata sulla mezzanotte.

ROUND (form 2, numeri)

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CEIL; FLOOR; NUMERI, FUNZIONI; ROUND (DATE); TRUNC; Capitolo 7

SINTASSI ROUND(*valore,precisione*)

DESCRIZIONE ROUND arrotonda un *valore* alla *precisione* desiderata. *precisione* è un intero positivo, negativo o nullo. Un numero negativo arrotonda il valore al numero di cifre fornito a sinistra della virgola. Un numero positivo arrotonda il valore al numero di cifre a destra della virgola.

ESEMPIO `ROUND(123.456,2) = 123.46`
`ROUND(123.456,0) = 123`
`ROUND(123.456,-2) = 100`
`ROUND(-123.456,2) = -123.46`

ROWID

TIPO Pseudocolonna

PRODOTTI Tutti

VEDERE ANCHE CHARTOROWID, PSEUDOCOLONNE, ROWIDTOCHAR, Capitolo 9

SINTASSI AAAAsYABQAAAAGzAAA

DESCRIZIONE Il ROWID è l'indirizzo logico di una riga ed è pertanto univoco all'interno del database. I ROWID possono essere selezionati o utilizzati nelle clausole *where* ma non possono essere modificati per mezzo di inserimenti, aggiornamenti o cancellazioni.

Non si tratta di un dato facente parte della colonna; è semplicemente un indirizzo logico creato sulla base di informazioni interne al database. Il ROWID è utilissimo all'interno di clausole *where* per aggiornare o cancellare rapidamente delle righe.

Il ROWID può variare se la tabella che contiene le righe viene esportata e reimportata. Questo strumento deve quindi essere utilizzato con la massima cautela.

ROWIDTOCHAR

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE CHARTOROWID, FUNZIONI DI CONVERSIONE

SINTASSI ROWIDTOCHAR(*RowId*)

DESCRIZIONE RowIdToCHAR trasforma un rowid interno di ORACLE in una stringa di caratteri. ORACLE effettua automaticamente questo tipo di conversione per cui questa funzione non è realmente necessaria.

ROWNUM

TIPO Pseudocolonna

PRODOTTI Tutti

VEDERE ANCHE PSEUDOCOLONNE, Capitolo 16

SINTASSI ROWNUM

DESCRIZIONE ROWNUM rende il numero di sequenza con il quale una riga è stata restituita dopo un select su una tabella. Per la prima riga ROWNUM vale 1, per il secondo 2 e così via. Si noti, comunque, che anche un semplice order by mette in disordine i ROWNUM, in quanto questi vengono assegnati alle righe prima che abbia luogo l'ordinamento. Vedere il Capitolo 16 per una discussione del problema.

RPAD

TIPO Funzione SQL

PRODOTTI Tutti, PL/SQL

VEDERE ANCHE CARATTERI, FUNZIONI; LPAD, LTRIM, RTRIM, Capitolo 6

SINTASSI RPAD(*stringa*,*lunghezza* [, 'set'])

DESCRIZIONE RPAD rende una stringa di una certa lunghezza aggiungendo un opportuno *set* di caratteri sulla sua destra. Se non viene specificato alcun *set* viene utilizzato il carattere di riempimento di default che è lo spazio.

ESEMPIO select RPAD('HELLO ',24,'WORLD') from DUAL;

produce il risultato seguente:

```
HELLO WORLDWORLDWORLDWOR
```

mentre:

```
select RPAD('CAROLYN',15,'-') from DUAL;
```

produce:

```
CAROLYN-----
```

RTRIM

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; LPAD, LTRIM, RPAD, Capitolo 6

SINTASSI RTRIM(*stringa* [, 'set'])

DESCRIZIONE RTRIM elimina tutte le ricorrenze di un *set* di caratteri a partire dal lato destro di una *stringa*.

ESEMPIO RTRIM('GEORGE','OGRE')

non produce *nulla*, una stringa NULL vuota di lunghezza zero! Invece:

```
RTRIM('EDYTHE','HET')
```

produce:

```
EDY
```

RUN

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE /,@,@@, EDIT, START

SINTASSI R[UN]

DESCRIZIONE RUN visualizza il comando SQL eventualmente nel buffer SQL e successivamente lo esegue. RUN è simile al comando / tranne che per il fatto che / non visualizza preventivamente il comando SQL.

SALVATAGGIO A DUE STADI

ORACLE gestisce le transazioni distribuite mediante una funzione speciale chiamata *salvataggio a due stadi*. Questo meccanismo garantisce che le transazioni risultino valide su tutti i nodi su cui avviene il salvataggio. Nel caso ci siano dei problemi

viene effettuato un rollback. Tutti i nodi salvano o annullano contemporaneamente, indipendentemente da quanto accade sulla rete. Il salvataggio a due stadi è un meccanismo attivo per default.

SAVE

PRODOTTI SQL*PLUS

VEDERE ANCHE EDIT, GET, Capitolo 5

SINTASSI SAV[E] *file[.ext]* [CRE[ATE] | REP[LACE] | APP[END]]

DESCRIZIONE SAVE salva il contenuto del buffer corrente in un file host di nome *file*. Se non viene specificata alcuna estensione, viene utilizzata l'estensione SQL. Questa estensione di default può essere modificata tramite l'istruzione 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 invece sovrascrive qualsiasi file preesistente e ne crea di nuovi all'occorrenza.

L'opzione APPEND aggiunge questo file al fondo di un file preesistente o ne crea uno nuovo se non esiste.

ESEMPIO La riga seguente salva il contenuto del buffer corrente all'interno di un file di nome lowcost.sql:

```
save lowcost
```

SAVEPOINT

TIPO Comando SQL

PRODOTTI Tutti, PL/SQL

VEDERE ANCHE COMMIT, ROLLBACK, SET TRANSACTION

SINTASSI SAVEPOINT *savepoint*

```
EXEC SQL SAVEPOINT [AT {database | :variabile }] savepoint
```

DESCRIZIONE Un SAVEPOINT è un punto interno a una transazione da cui è possibile effettuare un rollback. Vedere ROLLBACK.

La prima versione è tipica di SQL*PLUS mentre la seconda è tipica dell'SQL interno e permette di identificare il database su cui è stato creato il SAVEPOINT sia mediante un nome che attraverso una variabile. Lo scopo dei SAVEPOINT è quello di assegnare un nome a un gruppo di istruzioni SQL e, qualora sia necessario, annullare l'effetto della loro esecuzione. Le istruzioni possono essere annidate o sequenziali in modo da fornire un vasto controllo nella gestione degli errori. L'idea di base è quella di creare delle transazioni logiche che vengono salvate solo quando tutti i componenti sono stati eseguiti con successo. Se un passaggio provoca degli errori, si potranno fare delle modifiche e riprovare la transazione senza dover disfare nulla. Di seguito è riportato un esempio di struttura, evidenziata logicamente nello pseudocodice:

```
inizio della transazione logica
```

```
    savepoint ALPHA  
    istruzione SQL 1  
    funzione a  
    funzione b
```

```

if (condizione) then rollback al savepoint ALPHA

    savepoint BETA
    funzione c
    istruzione SQL 2
    istruzione SQL 3

    if (condizione) then rollback al savepoint BETA

    if (condizione) then rollback al savepoint ALPHA

        savepoint GAMMA
        istruzione SQL 4

        if (condizione) then rollback al savepoint GAMMA
        if (condizione) then rollback al savepoint BETA
        if (condizione) then rollback al savepoint ALPHA
        if (condizione) then commit oppure rollback

```

Questo mostra come gruppi di istruzioni SQL possano essere annullate assieme o singolarmente. In un programma host, oppure utilizzando PL/SQL o SQL*PLUS, è possibile che si vogliano impostare dei SAVEPOINT sulla base del risultato dell'ultima istruzione SQL o dell'ultima condizione controllata. ROLLBACK permette di annullare le operazioni effettuate a partire dall'ultimo SAVEPOINT.

Se una funzione, un controllo logico o un'istruzione SQL ha dei problemi, si può ripristinare lo stato precedente del database.

I nomi dei SAVEPOINT devono essere unici all'interno di ogni transazione (che termina dopo un COMMIT o un ROLLBACK incondizionato) ma non necessariamente in tutto il database. Ad esempio, è possibile assegnare gli stessi nomi a un SAVEPOINT e a una tabella (potrebbe rendere il codice più comprensibile utilizzare per il SAVEPOINT lo stesso nome della tabella da aggiornare). I nomi dei SAVEPOINT seguono le solite note regole di nomenclatura.

Se si fornisce a un SAVEPOINT un nome già utilizzato precedentemente, il nuovo sostituisce quello vecchio: il primo viene perso definitivamente e non c'è alcun modo di effettuare un rollback da quel punto.

Il parametro SAVEPOINT del file init.ora determina il numero massimo di SAVEPOINT possibili in una transazione. L'impostazione di default è 5, ma è possibile portarla fino a 255.

Una volta effettuato un COMMIT o un ROLLBACK incondizionato, tutti i SAVEPOINT precedenti vengono cancellati. Si ricordi che tutte le istruzioni DDL (come DISCONNECT, CREATE TABLE, CREATE INDEX e così via) effettuano 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 da ConText per valutare quanto una stringa rispetti le regole di ricerca. È possibile visualizzare il “punteggio” di una ricerca indi-

viduale; più spesso il punteggio viene confrontato con un valore di soglia. Vedere il Capitolo 29 per alcuni esempi di ricerca ConText e visualizzazione di punteggi.

SEGMENTO

Il segmento è un'altra unità di misura dello spazio allocato a una tabella, a un indice o a un cluster.

SEGMENTO DI ROLLBACK

Un segmento di rollback è quella parte di un tablespace che contiene le informazioni relative alle transazioni in modo da garantire l'integrità dei dati e renderli coerenti.

SEGMENTO TEMPORANEO

Si dice segmento temporaneo l'area di una tablespace utilizzata per contenere i risultati intermedi di un'istruzione SQL. Ad esempio, i segmenti temporanei vengono utilizzati nel riordino di tabelle molto estese.

SELECT (forma 1, SQL)

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE COLLATION, CONNECT BY, DELETE, DUAL, FROM, GROUP BY, HAVING, INSERT, JOIN, OPERATORI LOGICI, ORDER BY, OPERATORI DI QUERY, SOTTOQUERY, OPERATORI SINTATTICI, UPDATE, WHERE

SINTASSI

```

SELECT [DISTINCT | ALL]
      { *
      | { [utente.] {tabella | vista | snapshot} .*
          | espressione [ [AS] alias_colonna] }
      [, { [utente.] {tabella | vista | snapshot} .*
          | espressione [ [AS] alias_colonna] } ] ...
      FROM { [utente.] {tabella [PARTITION (nome_partizione)
                                | @dblink]
                           | [vista |snapshot] [@dblink] } [alias_tabella]
                           | [THE] ( sottoquery ) [alias_tabella]
                           | TABLE (colonna_tabella_annidata) [alias_tabella] }
      [, { [utente.] {tabella [PARTITION (nome_partizione) |
                                @dblink]
                           | [vista |snapshot] [@dblink] } [alias_tabella]
                           | [THE] ( sottoquery ) [alias_tabella]
                           | TABLE (colonna_tabella_annidata) } ] ... [WHERE
                                             condizione]
      [ [ START WITH condizione] CONNECT BY condizione
      | GROUP BY espressione [, espressione] ... [HAVING
                                                condizione] ] ...
      [ { UNION
          | UNION ALL
          | INTERSECT
          | MINUS} SELECT comando]
      [ ORDER BY {espressione | posizione | alias_colonna} [ASC | DESC]
```

```
[, {espressione | posizione | alias_colonna} [ASC | DESC] ] ...
| FOR UPDATE [OF [ [utente.] {tabella. | vista.} ] colonna
[, [ [utente.] {tabella. | vista.} ] colonna] ...] [NOWAIT]
```

DESCRIZIONE SELECT preleva delle righe da una o più tabelle (o viste, o snapshot) sia come comando a sé stante sia come sottoquery all'interno di altri comandi SQL (nella creazione di restrizioni e limitazioni) compresi select, insert, update e delete. ALL impone che vengano restituite tutte le righe che soddisfano le condizioni specificate (ed è l'impostazione di default). DISTINCT seleziona solo righe uniche: gli eventuali doppioni vengono ignorati.

Specificando un asterisco (da solo) si ottiene la visualizzazione di tutte le colonne di tutte le tabelle della clausola from. *tabella.** significa che verranno visualizzate tutte le colonne della tabella e può essere seguito da altri nomi di colonna ed espressioni, inclusa ROWID.

espressione può essere qualsiasi tipo di colonna. Può trattarsi del nome di una colonna, di un'espressione matematica, di un letterale o di una combinazione di funzioni. In genere si tratta di 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 una espressione. L'alias di colonna può essere opzionalmente preceduto dalla parola chiave AS. L'alias diviene l'intestazione di colonna nella visualizzazione e può essere gestito mediante i comandi SQL*PLUS column, ttitle e btitle. Può inoltre essere utilizzato nella clausola order by. Se si tratta di più di una parola o se contiene caratteri non visualizzabili deve essere racchiuso tra apici doppi.

utente, *tabella* e *dblink* denotano la tabella o la vista da cui prelevare le righe; sono opzionali, in caso di loro assenza ORACLE suppone si tratti dell'utente stesso e della connessione corrente. Specificando comunque il nome utente in una query si può ridurre l'entità del lavoro di ORACLE e rendere la query più rapida. Può essere specificata una sottoquery nella clausola FROM. ORACLE esegue la sottoquery e le righe risultanti vengono trattate come se provenissero da una vista.

L'alias di tabella in questo caso rinomina la tabella solo per la query corrente. A differenza degli alias delle espressioni, può essere impiegato ovunque all'interno dell'istruzione select (ad esempio nei prefissi di colonna). 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 vengono congiunti dei nomi molto lunghi per l'esecuzione di query. Inoltre gli alias vengono utilizzati nelle sottoquery correlate quando sia necessario unire una tabella a se stessa.

condizione può essere qualsiasi espressione logica. Può contenere funzioni, colonne o letterali. *posizione* consente alla clausola order by di identificare le espressioni sulla base della loro posizione relativa nella clausola select piuttosto che utilizzandone i nomi.

Si tratta di una caratteristica insostituibile quando si hanno espressioni complicate o si devono impiegare gli operatori UNION, INTERSECT e MINUS. Si sconsiglia l'uso delle posizioni nelle clausole order by (tranne quelle che UNION, INTERSECT e MINUS); al loro posto possono essere utilizzati gli alias. 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 se si tratta di un ordinamento ascendente o discendente. *colonna* è il nome di una colonna della tabella della clausola from; non può essere

sostituita da una espressione. NOWAIT significa che ogni eventuale select for update che incontri una riga bloccata viene arrestato e il controllo torna immediatamente all'utente invece di aspettare lo sblocco della risorsa.

La clausola for update of blocca le righe selezionate. select...for update of dovrebbe essere seguito immediatamente da un comando update...where o, se si decide di non aggiornare nulla, da un COMMIT o un ROLLBACK. Una volta bloccata una riga, gli altri utenti non sono in grado di modificarla fino a che non la si liberi con un comando COMMIT (o AUTOCOMMIT) o ROLLBACK. select...for update of non può contenere DISTINCT, GROUP BY, UNION, INTERSECT, MINUS o qualsiasi funzione di gruppo come MIN, MAX, AVG e 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 impedisce alcuna clausola of in cui elencare le tabelle ORACLE blocca tutte le tabelle della clausola from. Tutte le tabelle devono trovarsi nello stesso database e se sono presenti dei riferimenti a colonne LONG o a sequenze particolari, le tabelle devono trovarsi nello stesso database della colonna LONG o della sequenza.

Se si interrogano le colonne di una tabella annidata diventa necessaria la funzione THE che si occupa di "appiattire" la compenetrazione. Vedere il Capitolo 26 per degli esempi dettagliati.

Se si interroga una tabella partizionata è possibile indicare il nome della partizione all'interno della clausola from della query. Vedere il Capitolo 17.

Le altre clausole dell'istruzione select sono descritte sotto il loro nome in questo stesso capitolo.

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, interno SQL)

TIPO Comando SQL interno

PRODOTTI Precompilatori

VEDERE ANCHE CONNECT, DECLARE CURSOR, DECLARE DATABASE, EXECUTE, FETCH, FOR, PREPARE, UPDATE (forma 2), WHENEVER

SINTASSI EXEC SQL [AT {database!,:variabile}]

```

SELECT elenco_select
      INTO :variabile [,:variabile]...
      FROM elenco_tabelle
      [ WHERE condizione ]
      [ CONNECT BY condizione [START WITH condizione] ]
      [ GROUP BY espressione [,espressione]... ] [HAVING condizione]
      [ { UNION [ALL] | INTERSECT | MINUS } SELECT ... ]
      [ ORDER BY {espressione | posizione} [ASC|DESC] ...
          [ ,espressione | posizione} [ASC|DESC] ]...
      [ FOR UPDATE [OF elenco_colonne] [NOWAIT] ]
```

DESCRIZIONE Vedere la descrizione delle varie clausole nella forma 1 di SELECT. Di seguito sono elencati gli elementi caratteristici della versione interna.

- AT *database*, assegna opzionalmente un nome a un database aperto precedentemente da CONNECT tra quelli dichiarati dall'istruzione DECLARE DATABASE.
- INTO :*variabile* [,*variabile*] 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 scalari (non array).
- La clausola select può riferirsi a variabili host in qualsiasi posizione nella quale avrebbe potuto essere utilizzata una costante.

Si può utilizzare la versione interna di SELECT con array variabili) per prelevare diverse righe con un unico FETCH. Inoltre è possibile utilizzare le istruzioni DECLARE CURSOR o PREPARE per un FETCH successivo specificando eventualmente la clausola for update of. L'ultima istruzione update può riferirsi alle colonne indicate nelle clausole for update of attraverso la sua 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 corrispondono al criterio di ricerca vengono bloccate alla loro apertura. COMMIT rilascia i blocchi e da quel momento non sono consentiti ulteriori FETCH. Questo significa che è indispensabile effettuare il FETCH e manipolare tutte le righe da aggiornare prima di impartire il comando COMMIT.

ESEMPIO

```
exec sql select Nome, Eta, Alloggio, Eta - :Pensione
               into :Impiegato, :Eta, :Domicilio, :Limite
               from LAVORATORE
               where Eta >= :Pensione;
```

SELECT...INTO

TIPO Versione PL/SQL di un'istruzione SQL

PRODOTTI PL/SQL

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 che si usa in DECLARE CURSOR. Questo formato utilizza il cursore implicito di nome SQL, esegue un blocco di istruzioni (racchiuso tra BEGIN ed END) e copia i valori da un'unica riga sotto forma di stringa di caratteri o record la cui struttura sia stata dichiarata da %ROWTYPE in modo che rispecchi le colonne selezionate. Se viene utilizzata la versione che restituisce i risultati all'interno di variabili, queste devono essere state dichiarate e devono avere tipi di dati compatibili.

Questo tipo di select può essere utilizzato in un ciclo in cui appaiano diverse variabili PL/SQL nella clausola where ma debba essere restituita una sola riga alla vol-

ta. Se ne viene prodotta più d'una scatta l'eccezione TOO_MANY_ROWS e SQL%ROWCOUNT varrà 2 (*vedere ECCEZIONI* per ulteriori dettagli). SQL%FOUND varrà TRUE.

Se non viene prodotta alcuna riga scatta l'eccezione NO_DATA_FOUND, SQL%ROWCOUNT viene impostato a 0 e SQL%FOUND a FALSE.

ESEMPIO DECLARE

```
    IMPIEGATO      LAVORATORE%ROWTYPE;
    ...
BEGIN
    select Nome, Eta, Alloggio into IMPIEGATO
        from LAVORATORE
       where Lavoratore = 'BART SARJEANT';
```

o in alternativa:

```
DECLARE
    Chi      CHAR(25);
    Anni    NUMBER;
    Casa    CHAR(25);
    ...
BEGIN
    select Nome, Eta, Alloggio into Chi, Anni, Casa
        from LAVORATORE
       where Lavoratore = 'BART SARJEANT';
```

SEQUENZA

Una sequenza è un oggetto di database utilizzato per generare numeri interi unici da impiegare come chiavi primarie. *Vedere CREATE SEQUENCE*.

SERVER MANAGER

Server Manager è l'utilità di ORACLE utilizzata dai DBA durante la gestione e il monitoraggio del database.

SESSIONE

Una sessione è quella serie di eventi che avviene dalla connessione di un utente a SQL alla sua successiva disconnessione.

SET

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE SET TRANSACTION, SHOW, Capitolo 5

SINTASSI SET *caratteristica* *valore*

dove *caratteristica* e *valore* appartengono all'elenco seguente. Il valore di default è quello riportato per primo.

```
APPI[NFO] {ON|OFF|testo}
ARRAY[SIZE] {20|intero}
AUTO[COMMIT] {ON|OFF|IMMEDIATE|n}
AUTOP[RINT] {ON|OFF}
AUTOT[RACE] {ON|OFF|TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]]
BLOCKTERMINATOR {.|simbolo}BUF[FER] buffer
```

CLOSECUR[SOR] {ON|OFF}
 CMDS[EP] {OFF|ON};*simbolo*
 COLSEP {_*testo*}
 COM[PATIBILITY] {V6|V7|V8|NATIVE}
 CONCAT {.;*simbolo*|ON|OFF}
 COPYCOMMIT {0;*intero*}
 COPYTYPECHECK {ON|OFF}
 DCL[SEP] {!;*simbolo*}
 DEF[INE] {&;*simbolo*;OFF;ON}
 DOC[UMENT] {OFF;ON}
 ECHO {OFF;ON}
 EDITF[ILE] [*nomefile*[.ext]]
 EMBEDDED {OFF;ON}
 ESCAPE {\;*simbolo*;OFF;ON}
 FEED[BACK] {6;*intero*;OFF;ON}
 FLAGGER {OFF;ENTRY;INTERMED[IATE];FULL}
 FLUSH {OFF;ON}
 HEA[DING] {OFF;ON}
 HEADS[EP] {||*simbolo*;OFF;ON}
 LIN[ESENCE] {80;*intero*}
 LONG {80;*intero*}
 LONGC[HUNKSIZE] {80;n}
 MAXD[ATA] *intero*
 NEWP[AGE] {1;*intero*}
 NULL testo NUMF[ORMAT] *formato*
 NUM[WIDTH] {10;*intero*}
 PAGES[IZE] {14;*intero*}
 PAU[SE] {OFF;ON;*testo*}
 RECSEP {WR[APPED];EA[CH];OFF}
 RECSEPCHAR {_*simbolo*}
 SCAN {ON|OFF}
 SERVEROUT[PUT] {ON|OFF} [SIZE *n*] [FOR[MAT]
 {WRA[PPED] | WOR[D_WRAPPED] | TRU[NCATED]}]
 SHOW[MODE] {OFF;ON}
 SPA[CE] {1;*intero*}
 SQLC[ASE] {MIX[ED];LO[WER];UP[PER]}
 SQLC[ONTINUE] {'>';*testo*}
 SQLN[NUMBER] {ON|OFF}
 SQLPRE[FIX] {#;*simbolo*}
 SQLP[ROMPT] {SQL>;*testo*}
 SQLT[ERMINATOR] {;;*simbolo*;OFF;ON}
 SUF[FIX] {SQL;*testo*}
 TAB {OFF;ON}
 TERM[OUT] {ON|OFF}
 TI[ME] {OFF;ON}
 TIMI[NG] {OFF;ON}
 TRIM[OUT] {ON|OFF}
 TRIMS[POOL] {ON|OFF}
 TRU[NCATE] {OFF;ON}
 UND[ERLINE] {-;*carattere*;ON|OFF}
 VER[IFY] {ON|OFF}
 WRAP {ON|OFF}

DESCRIZIONE SET imposta una caratteristica di SQL*PLUS, attivandola, disattivandola o assegnandole un certo valore. Tutte queste caratteristiche vengono visualizzate per mezzo del comando SHOW. Se si indica il nome di un comando, SHOW mostra solo quello specificato; altrimenti li visualizza tutti; poiché sono in numero elevato è pratico reindirizzare l'output su un file di spool, digitare SHOW e poi spool off. In questo modo tutte le opzioni e il loro stato corrente vengono riversati in un file facilmente importabile nel proprio editor preferito.

Nelle caratteristiche seguenti, il valore di default viene indicato per primo.

APP[INFO] {ON|OFF|*testo*} gestisce la registrazione di file di comandi attraverso il pacchetto DBMS_APPLICATION_INFO. Il nome registrato compare nella colonna Module della colonna di prestazioni dinamiche V\$SESSION.

ARRAY[SIZE] {20|*n*} imposta la dimensione dei pacchetti di righe che SQL*PLUS carica a ogni prelievo. L'intervallo spazia da 1 a 5000. Valori più elevati migliorano l'efficienza di quelle query in cui ha senso prelevare molte righe ma utilizzano molta memoria. Valori superiori a 100 raramente producono miglioramenti prestazionali sensibili. ARRAYSIZE non ha alcun altro effetto su SQL*PLUS.

AUTO[COMMIT] {OFF|ON|IMM|*n*} ON o IMM fanno sì che al completamento di ciascuna istruzione SQL vengano salvati tutte le modifiche al database. *n* imposta il numero di istruzioni dopo cui effettuare il salvataggio. OFF disabilita il salvataggio automatico obbligando, di fatto, a indicare esplicitamente le richieste di salvataggio per mezzo del comando COMMIT. Molti comandi, come QUIT, EXIT, CONNECT e tutte le istruzioni DDL) effettuano esse stesse un COMMIT di tutte le modifiche ancora pendenti.

AUTOP[RINT] {ON|OFF} decide se SQL*Plus debba visualizzare automaticamente le variabili mute utilizzate nei blocchi PL/SQL o nelle procedure.

AUTOT[RACE] {ON|OFF|TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]] permette di visualizzare il percorso di esecuzione di una query dopo che si è conclusa. Per far ciò è necessario disporre di una tabella di nome PLAN_TABLE all'interno del proprio schema (può essere creata avviando il file UTLXPLAN.SQL all'interno della directory /rdbms/admin).

BLOCKTERMINATOR {.*simbolo*} imposta il simbolo che dev'essere utilizzato come terminatore di un blocco PL/SQL. Non può essere né una lettera né un numero. Per eseguire un blocco utilizzare i comandi RUN o / (slash).

BUF[FER] {*buffer*} imposta il buffer corrente. All'avvio di SQL*PLUS il buffer di default è quello di SQL.

CLOSECUR[SOR] {ON|OFF} decide se chiudere e riaprire i cursori dopo ogni istruzione SQL.

CMDS[EP] {OFF|ON|;.*simbolo*} imposta il simbolo da utilizzare per separare i comandi eventualmente impartiti assieme su un'unica riga. ON e OFF decidono se consentire o meno la presenza di più comandi su un'unica riga.

COLSEP {.*testo*} imposta la sequenza di caratteri da stampare tra le colonne. Vedere SET RECSEP.

COM[PATIBILITY] {V6|V7|V8|NATIVE} specifica la versione di ORACLE a cui si è connessi. Se ad esempio si impone il parametro di compatibilità su V6, i tipi CHAR vengono trattati come colonne di lunghezza variabile; se si specifica V7, i tipi CHAR diventano colonne a lunghezza fissa. Se si sceglie NATIVE, il database determina automaticamente la versione di ORACLE cui si è collegati.

CONCAT {.:*simbolo*} imposta il simbolo terminatore o delimitatore di una variabile utente seguita da un simbolo, un carattere o una parola che potrebbe altrimenti essere interpretata come parte del nome della variabile.

CONCAT {ON|OFF} Impostando CONCAT a ON il suo valore torna a ‘.:’.

COPYCOMMIT {0|*n*} Il comando COPY salva le righe sul database di destinazione *n* righe alla volta. I valori possibili variano tra 0 e 5000. Se valore vale 0 il COMMIT viene effettuato solo al termine dell'operazione di copia.

COPYTYPECHECK {ON|OFF} permette di sopprimere i controlli di tipo durante l'uso del comando COPY.

DCL[SEP] {!:*simbolo*} imposta il simbolo utilizzato per separare comandi di sistema operativo introdotti sulla stessa riga SQL*PLUS. La riga seguente visualizza “Ciao a tutti” sul terminale dell'utente:

```
host echo Ciao a tutti
```

Questa riga:

```
host echo Ciao a tutti!echo Ciao a tutti!echo Ciao a tutti
```

visualizza:

```
Ciao a tutti
Ciao a tutti
Ciao a tutti
```

Il separatore di default è ! e funziona solo assieme a comandi HOST o \$. DCL è il linguaggio operativo di VMS di Digital Equipment.

DEF[INE] {&!*simbolo*} definisce il carattere utilizzato come prefisso per una variabile di sostituzione.

DEF[INE] {ON|OFF} decide se SQL*PLUS debba cercare e sostituire le variabili di sostituzione e se debba caricarle assieme ai loro valori.

DOC[UMENT] {OFF|ON} ON permette il funzionamento del comando DOCUMENT. OFF fa sì che SQL*PLUS lo ignori, di fatto provocando l'interpretazione delle righe seguenti come istruzioni SQL*PLUS o SQL. Vedere DOCUMENT per ulteriori dettagli.

ECHO {OFF|ON} ON visualizza i comandi durante l'esecuzione di un file di avvio. OFF effettua un'esecuzione nascosta. L'output dei comandi è comunque controllato da TERMOUT.

EDITF[ILE] [*nomefile*[.*ext*]] imposta il nome di default dei file creati dal comando EDIT.

EMBEDDED {OFF|ON} ON permette a un nuovo rapporto di una serie di cominciare in qualsiasi punto di una pagina. OFF forza il nuovo rapporto su una nuova pagina.

ESCAPE {!\i simbolo} definisce il carattere di ESCAPE da utilizzare prima del simbolo di DEFINE in modo che possa essere visualizzato invece di essere interpretato come un nome di variabile.

ESCAPE {OFF|ON} OFF disabilita il carattere di ESCAPE. ON abilita il carattere di ESCAPE di default (la barra inversa “\”). Ad esempio, se il carattere di ESCAPE è \, la riga seguente:

```
ACCEPT Report prompt 'Inserire P\&L nome report:'
```

mostra:

Inserire P&L nome report:

e &L non viene trattato come nome di variabile.

FEED[BACK] {6|n|OFF|ON} decide se SQL*PLUS debba visualizzare “records selected” dopo la selezione di almeno *n* record. ON e OFF abilitano e disabilitano questa funzione. SET FEEDBACK 0 ha lo stesso effetto di FEED OFF.

FLAGGER {OFF|ENTRY|INTERMED|IATE|FULL} imposta il livello FIPS per la compatibilità SQL92.

FLUSH {ON|OFF} OFF viene utilizzato quando un file di avvio può essere eseguito senza produrre alcun messaggio e senza richiedere alcun intervento dell’utente fino al suo completamento. OFF permette al sistema operativo di evitare di inviare l’output sul video. ON riabilita l’output sul terminale. OFF può in taluni casi migliorare le prestazioni.

HEA[DING] {ON|OFF} OFF elimina le intestazioni che appaiono normalmente sopra le colonne. ON ne riabilita la visualizzazione.

HEADS[EP] {; | simbolo} *simbolo* è il carattere di separazione delle intestazioni. L’impostazione di default è la pipe (la barra verticale continua o discontinua).

HEADS[EP] {ON|OFF} ON e OFF attivano e disattivano l’opzione. Se è disattivata il simbolo di separazione viene stampato come qualsiasi altro carattere.

LIN[ESIZE] {80|*n*} imposta la larghezza della riga, il numero totale di caratteri visualizzabili su una riga prima di passare a quella successiva. Questo numero viene utilizzato da SQL*PLUS nel calcolo delle centratore e delle giustificazioni dei titoli. Il valore massimo è 999.

LONG {80|*n*} imposta la larghezza massima per la copia di valori LONG. *n* può essere compreso tra 1 e 32767 ma deve essere minore del valore di MAXDATA.

LONGC[HUNKSIZE] {80|*n*} imposta la dimensione, in caratteri, degli blocchi attraverso i quali SQL*Plus preleva un valore LONG.

MAXD[ATA] *n* imposta la larghezza di riga massima che SQL*PLUS è in grado di gestire. I valori di default e quelli massimi di questa impostazione variano da un sistema operativo all’altro. Vedere l’*ORACLE Installation and User’s Guide* per ulteriori dettagli.

NEWP[AGE] {1|*n*} imposta il numero di righe vuote da stampare tra il fondo di una pagina e il titolo iniziale della seguente. Uno 0 (zero) invia un avanzamento foglio (form feed) all’inizio di ogni pagina. Sul video questo corrisponde generalmente alla cancellazione dello schermo.

NULL *testo* imposta una stringa che SQL*PLUS sostituisce eventuali valori NULL rilevati.

NUMF[ORMAT] *formato* imposta il formato numerico di default nella visualizzazione di numeri e dati. Vedere NUMERICI, FORMATI per dettagli.

NUM[WIDTH] {10|*n*} imposta la larghezza di default per la visualizzazione dei numeri.

PAGES[IZE] {14|*n*} imposta il numero di righe per pagina. Vedere il Capitolo 4 per l’uso di pagesize e newpage.

PAU[SE] {OFF|ON|*testo*} ON impone a SQL*PLUS di attendere la pressione del tasto INVIO dopo la visualizzazione di ogni pagina di output. OFF disabilita questa funzione. *testo* è il messaggio che SQL*PLUS visualizza al fondo dello schermo

mentre attende la pressione del 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 per mezzo di un carattere definito tramite RECSEPCHAR (*vedere anche SET*). Per default questo carattere è lo spazio e la funzione è attiva sui soli record (e quindi sulle colonne) che vanno a capo. Ad esempio, di seguito sono riportate le otto righe della colonna Citta; il formato di visualizzazione è troppo stretto per poter rappresentare una delle Citta:

```
column Citta format a9

select Citta from COMFORT;

CITTA
-----
SAN FRANC
ISCO

SAN FRANC
ISCO

SAN FRANC
ISCO

SAN FRANC
ISCO

KEENE
KEENE
KEENE
KEENE
```

Quando RECSEP è disattivato l'effetto è quello seguente:

```
set recsep off

select Citta from COMFORT;
CITTA
-----
SAN FRANC
ISCO
SAN FRANC
ISCO
SAN FRANC
ISCO
SAN FRANC
ISCO
KEENE
KEENE
KEENE
KEENE
```

RECSEPCHAR {‘ ‘ | simbolo} imposta il simbolo di separazione di RECSEP. Il simbolo di default è lo spazio.

SCAN {ON|OFF} La sostituzione delle variabili avviene a seguito della scansione delle istruzioni da parte di SQL*PLUS alla ricerca del simbolo di sostituzione. Se SCAN viene disattivato tutto questo processo si interrompe.

SERVEROUT[PUT] {ON|OFF} [SIZE n] [FOR[FORMAT] {WRA[PPED] : WOR[D_WWRAPPED] : TRU[NCATED]}] permette di visualizzare l'output delle procedure PL/SQL (dal package DBMS_OUTPUT, vedere l'*Application Developer's Guide*). WRAPPED, WORD_WWRAPPED, TRUNCATED e FORMAT determinano il tipo di formattazione dell'output.

SHOW[MODE] {OFF|ON} attiva la visualizzazione delle impostazioni precedenti e di quelle successive ad un comando SET. OFF disattiva entrambe.

SPA[CE] {1|n} imposta il numero di spazi visualizzati tra le colonne durante l'output di una colonna. Il valore massimo è 10.

SQLC[ASE] {MIX[ED] : LO[WER] : UP[PER]} converte tutto il testo, compresi letterali, e identificatori (anche contenuti all'interno di blocchi SQL o PL/SQL) in lettere maiuscole o minuscole prima di farlo eseguire da ORACLE. MIXED lascia tutto come lo si è digitato; LOWER converte tutto in lettere minuscole; UPPER converte tutto in maiuscolo.

SQLC[ONTINUE] {‘ > ‘ | testo} imposta la sequenza di caratteri da visualizzare come messaggio guida quando una riga di comando troppo lunga deve continuare sulla successiva. Se si inserisce un trattino (-) al fondo della riga di comando di SQL*PLUS, SQLCONTINUE visualizza il prompt sulla sinistra della riga seguente.

SQLN[NUMBER] {ON|OFF} attivando questa opzione tutte le eventuali righe SQL oltre la prima avranno dei numeri di riga come messaggi guida. Se la si disattiva, il prompt SQLPROMPT apparirà solo sulle righe aggiuntive.

SQLPRE[FIX] {#|simbolo} imposta il carattere prefisso SQL che provoca l'immediata esecuzione di un comando SQL*PLUS (come COLUMN), anche durante l'introduzione di comandi SQL o PL/SQL. Il comando SQL*PLUS viene eseguito immediatamente.

SQLP[ROMPT] { SQL>|testo} imposta il prompt di SQL visualizzato quando SQL-NUMBER è OFF.

SQLT[ERMINATOR] {;|simbolo} imposta il simbolo utilizzato per terminare i comandi SQL e per cominciarne l'immediata esecuzione. OFF significa che nessun simbolo SQL verrà riconosciuto come terminatore e dunque l'utente dovrà terminare i comandi con una riga vuota dopo l'ultima riga di codice.

SQLT[ERMINATOR] {ON|OFF} ON reimposta il carattere al simbolo di default (“;”), indipendentemente da eventuali impostazioni precedenti.

SUFFIX {SQL|testo} imposta l'estensione (il tipo di file) di default che SQL*PLUS aggiunge ai file salvati. Vedere SAVE.

TAB {OFF|ON} disattivando l'opzione si forza SQL a utilizzare solo spazi nella formattazione di colonne e rapporti testuali. Il valore di default di TAB varia da sistema a sistema. Per visualizzarlo si usi SHOW TAB. Impostandola si forza SQL ad utilizzare i caratteri di tabulazione al posto degli spazi.

TERM[OUT] {ON|OFF} questa opzione visualizza l'output dei file di avvio di SQL*PLUS. Se la si imposta a OFF è possibile riscontrare miglioramenti prestazionali.

TI[ME] {OFF|ON} quando TIME è attivo, SQL*PLUS visualizza l'ora corrente (dell'orologio di sistema) prima di ciascun prompt. OFF disattiva questa opzione.

TIM[ING] {OFF|ON} questa opzione visualizza il tempo impiegato dall'esecuzione di ogni comando SQL (*vedere TIMING* per verificare come un comando con lo stesso nome operi in maniera completamente diversa).

TRIM[OUT] {ON|OFF} per funzionare correttamente in precedenza dev'essere stato impartito il comando SET TAB ON. Quando TRIM è attivo le righe vuote al termine di ciascuna riga visualizzata vengono soppresse cosa che spesso comporta significativi miglioramenti prestazionali, specialmente su linee commutate. OFF continua a visualizzare le righe vuote. TRIMOUT ON non influenza l'output reindirizzato su file.

TRIMS[POOL] {ON|OFF} rimuove gli spazi al termine di ogni riga. OFF disattiva l'impostazione.

TRU[NCATE] {OFF|ON} Al posto di TRUNCATE, comando obsoleto, è meglio utilizzare SET WRAP OFF. TRUNCATE taglia una parte di colonna in modo che quella rimanente rientri nel margine stabilito dal comando COLUMN FORMAT. SET WRAP OFF dà gli stessi risultati di SET TRUNCATE ON ma è preferibile, dato che il comando SHOW è in grado di visualizzarne l'impostazione, cosa non più possibile con TRUNCATE.

UND[ERLINE] {-|carattere} imposta il carattere di sottolineatura per le intestazioni delle colonne. Il carattere di default è il trattino (_).

UND[ERLINE] {ON|OFF} attiva o disattiva la sottolineatura senza modificare il carattere.

VER[IFY] {ON|OFF} questa impostazione obbliga SQL*PLUS a visualizzare i vecchi e i nuovi valori delle variabili prima di eseguire il codice SQL in cui si trovano. OFF disattiva la visualizzazione.

WRAP {ON|OFF} questa impostazione divide una riga su due linee se la sua lunghezza è superiore a quella impostata in COLUMN. OFF tronca la parte destra di una colonna se è troppo lunga per entrare tutta in una linea. Si usino le clausole wrapped del comando COLUMN per modificare l'impostazione di WRAP su alcune particolari colonne.

SET CONDITION

Una set condition è un'espressione logica contenente una query. Ad esempio "Nome IN (select...)". Il suo nome deriva dal fatto che la condizione di query produrrà un set di record.

SET CONSTRAINTS

TIPO Comando SQL

PRODOTTI Tutti, PL/SQL

VEDERE ANCHE VINCOLO DI INTEGRITÀ

SINTASSI SET CONSTRAINT[S] { *vincolo* [, *vincolo*] ... | ALL }
{ IMMEDIATE | DEFERRED }

DESCRIZIONE SET CONSTRAINTS DEFERRED ordina a ORACLE di non controllare la validità di un vincolo fino al salvataggio della transazione. Questo permette di ritardare il controllo di integrità dei dati, utile quando si effettuano DML su diverse tabelle e non si può garantire un ordine alle transazioni. Per effettuare queste

operazioni si deve essere in possesso del privilegio SELECT o essere i legittimi proprietari della tabella.

L'impostazione di default è SET CONSTRAINTS IMMEDIATE: i controlli di integrità vengono effettuati non appena viene inserito un nuovo record nella tabella.

SET ROLE

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE ALTER USER, CREATE ROLE, Capitolo 18

SINTASSI SET ROLE { ruolo [IDENTIFIED BY *password*]
[, ruolo [IDENTIFIED BY *password*]]... |
ALL [EXCEPT ruolo[, ruolo]...] |
NONE }

DESCRIZIONE SET ROLE abilita o disabilita i ruoli assegnati a un utente nella sessione SQL corrente. La prima opzione permette di specificare dei ruoli fornendo opzionalmente una password (se il ruolo lo richiede, vedere CREATE ROLE). La seconda opzione permette di abilitare tutti (ALL) i ruoli a eccezione di (EXCEPT) quelli indicati, che devono essere stati assegnati direttamente dallo stesso utente. L'opzione ALL non abilita i ruoli protetti da password. La terza opzione, NONE, disabilita tutti i ruoli della sessione corrente.

SET TRANSACTION

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE COMMIT, ROLLBACK, SAVEPOINT, TRANSAZIONE

SINTASSI SET TRANSACTION {READ ONLY | USE ROLLBACK SEGMENT *segmento*}

DESCRIZIONE SET TRANSACTION inizia una transazione. La transazione standard SQL garantisce già un'ottima coerenza dei dati in lettura. Alcune transazioni però richiedono garanzie ancor maggiori dato che diverse serie di istruzioni select, ciascuna all'opera su una o più tabelle, dovranno vedere dati coerenti gli uni con gli altri (ovvero dati che si riferiscono a uno stesso istante di tempo). SET TRANSACTION READ ONLY specifica questo meccanismo più protetto di gestione dei dati. Nessun comando (anche di altri utenti) può modificare i dati di un'area sulla quale vengono effettuate operazioni di select attraverso questi tipi di transazioni. 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 comandi quali insert, update o delete ORACLE assegna particolari segmenti di rollback. SET TRANSACTION USE ROLLBACK SEGMENT specifica che la transazione corrente utilizzerà uno specifico segmento di rollback.

```
select Nome, Direttore  
      from LAVORATORE, ALLOGGIO  
     where LAVORATORE.Alloggio = LODGING.Alloggio;
```

Le modifiche apportate a Direttore dopo l'esecuzione della query ma prima che siano state prelevate tutte le righe non appaiono tra i risultati. Se invece le due tabelle fossero state interrogate sequenzialmente come in:

```
select Nome, Alloggio from LAVORATORE;
select Direttore, Alloggio from ALLOGGIO;
```

il campo Direttore in ALLOGGIO avrebbe potuto essere modificato durante l'esecuzione della prima select e analogamente il campo Alloggio in LAVORATORE durante l'esecuzione della seconda select. Il tentativo di raggruppare i risultati di queste due query utilizzando istruzioni di programma (invece di usare il join di tabelle del primo esempio) produce risultati incoerenti. La soluzione è data da:

```
commit;
set transaction read only;
select Nome, Alloggio from LAVORATORE;

select Direttore, Alloggio from ALLOGGIO;
commit;
```

In questo modo i dati di entrambe le tabelle vengono congelati prima che le select prelevino le righe. La consistenza di questo metodo è identica a quella fornita dal metodo del join di tabelle. I dati vengono mantenuti costanti fino all'esecuzione di un COMMIT o di un ROLLBACK.

È molto importante utilizzare due commit (uno prima e uno dopo). SET TRANSACTION dev'essere la prima istruzione SQL di una transazione. Il COMMIT appena prima assicura che questo sia vero. Il COMMIT alla fine rilascia lo snapshot sui dati. È importante in quanto ORACLE deve reimpossessarsi delle risorse utilizzate per mantenere i dati costanti.

Se i dati congelati provengono da tabelle di dimensioni molto grandi, o se le transazioni sono molto lunghe in termini di tempo, il DBA può scegliere di creare segmenti di rollback addizionali per l'archiviazione degli snapshot.

SGA

Vedere System Global Area.

SHOW

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE SET

SINTASSI SHO[W] { *caratteristica* | ALL | BTI[TLE] | LNO | PNO | REL[EASE] |
 SPOO[L] | TTI[TLE] | USER | WRAP }

DESCRIZIONE SHOW visualizza il valore di una (o tutte) caratteristica di SET o degli altri oggetti SQL*PLUS. È possibile la visualizzazione di diversi oggetti. Ciascuna risposta viene fornita su una linea diversa. Le righe restituite vengono ordinate alfabeticamente.

BTI[TLE] visualizza la definizione corrente di btitle.

LNO mostra il numero di linea corrente (la linea della pagina in corso di visualizzazione).

PNO visualizza il numero della pagina corrente.

REL[EASE] fornisce il numero di versione di ORACLE.

SPOO[L] mostra la situazione di reindirizzamento dell'output su file (se è attivo o no). *Vedere* SPOOL.

SQLCODE mostra il codice dell'errore ORACLE più recente.
TTI[TLE] visualizza la definizione corrente di title.
USER mostra l'identificativo dell'utente.
WRAP mostra l'impostazione relativa all'avanzamento riga (TRUNCATE non funziona con SHOW).

SHOWMODE (SQL*PLUS)

Vedere SET.

SHUTDOWN

Questo termine indica la disconnessione di un'istanza da un database e il suo successivo spegnimento. Vedere anche START.

SIGN

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE +, PRECEDENZE, Capitolo 7

SINTASSI SIGN(*valore*)

DESCRIZIONE Restituisce 1 se il valore è positivo, -1 se è negativo, 0 se vale zero.

ESEMPIO SIGN(33) = 1

SIGN(-.6) = -1

SIGN(0) = 0

SIN

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE ACOS; ASIN; ATAN; ATAN2; COS; NUMERI, FUNZIONI; SINH; TAN; Capitolo 7

SINTASSI SIN(*valore*)

DESCRIZIONE SIN restituisce il seno dell'angolo il cui *valore* è espresso in radianti.

ESEMPIO select SIN(30*3.141593/180) Seno -- seno di 30 gradi in radianti
from DUAL;

SENO

.5

SINONIMO

Un sinonimo è un nome assegnato a una tabella o a una vista per semplificare i successivi riferimenti a quella stessa risorsa. Se si ha la possibilità di accedere a una tabella di proprietà 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 21 e 32.

SINONIMO PUBBLICO

Un sinonimo pubblico è un sinonimo di un oggetto che un utente ha reso accessibile a tutti attraverso il privilegio CREATE PUBLIC SYNONYM.

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).

SINH

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE ACOS; ASIN; ATAN; ATAN2; COS; NUMERI, FUNZIONI; SIN; TAN; Capitolo 7

SINTASSI SINH(*valore*)

DESCRIZIONE SINH restituisce il valore del seno iperbolico dell'angolo *valore*.

SISTEMA OPERATIVO

Un sistema operativo è un programma che gestisce le risorse di un computer e interagisce i programmi con l'hardware disponibile.

SISTEMA SERVER

Il sistema server è il sistema ORACLE che contiene il database cui utenti remoti possono accedere attraverso SQL*NET.

SISTEMA A PARTIZIONE CONDIVISA

Si tratta di configurazione di ORACLE in cui diversi sistemi ORACLE posti su nodi di rete diversi condividono gli stessi file e database. Ogni nodo è detto istanza.

SMON

Il System Monitor è uno dei processi in background di ORACLE; esso effettua la gestione dei segmenti temporanei inutilizzati. Vedere PROCESSI IN BACKGROUND.

SNAPSHOT

Uno snapshot è una copia locale di un blocco di dati remoto. Esso replica una parte di una tabella remota (o anche tutta) o replica il risultato di una query su tabelle diverse. Gli aggiornamenti dei dati replicati possono essere demandati automaticamente al database (a intervalli di tempo specificati dall'utente) o effettuati manualmente. Vedere il Capitolo 28, CREATE SNAPSHOT, CREATE SNAPSHOT LOG.

SOSTITUZIONE

Vedere &, &&, ACCEPT, DEFINE

SOTTOQUERY

TIPO Caratteristica del linguaggio SQL

PRODOTTI Tutti, SQL*PLUS

VEDERE ANCHE COPY, CREATE TABLE, DELETE, INSERT, SELECT, UPDATE, Capitolo 11

SINTASSI COPY [FROM *utente/password@database*]
[TO *utente/password@database*]

```

{APPEND | CREATE | INSERT | REPLACE}
table [ (colonna [,colonna]...) ]
      USING query;
CREATE TABLE tabella ...
      AS query;
DELETE FROM tabella [alias]
      [ WHERE ... (sottoquery)];
INSERT INTO tabella [ (colonna [,colonna]...) ]
      query;
SELECT [* | (espressione [,espressione]...)] 
      FROM tabella
      [ WHERE ... (sottoquery)];
UPDATE tabella [alias]
      SET (colonna [,colonna]...) = (sottoquery)
      [ WHERE condizione ];

```

DESCRIZIONE Una query (cioè un’istruzione di selezione) può essere utilizzata all’interno di altre istruzioni SQL (dette *istruzioni genitrici* o esterne), tra cui CREATE TABLE, delete, insert, select, update e la versione SQL*PLUS del comando COPY, onde definire le righe e le colonne da utilizzare nella propria esecuzione. I risultati della query figlia (detta anche *sottoquery*) non vengono visualizzati ma vengono passati all’istruzione esterna. Questo costrutto deve rispettare le regole seguenti:

- Nei comandi update e CREATE TABLE, la sottoquery deve ritornare un valore per ciascuna colonna da inserire o aggiornare. Il o i valori vengono quindi utilizzati dall’istruzione SQL esterna per inserire o aggiornare le righe.
- Una sottoquery non può contenere le clausole order by e for update of.
- Nella clausola where dell’istruzione select viene utilizzata una sottoquery “correlata” che si riferisce all’alias della tabella utilizzata dal comando di selezione esterno. Vedere il Capitolo 11 per ulteriori dettagli.

A parte le restrizioni appena citate, una sottoquery obbedisce alle stesse leggi che governano le normali istruzioni select.

SOUNDEX

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE LIKE, Capitoli 6 e 29

SINTASSI SOUNDEX(*stringa*)

DESCRIZIONE SOUNDEX trova tutte le parole che somigliano a quella fornita. SOUNDEX opera alcune assunzioni sulla usuale pronuncia di combinazioni di lettere tipiche. Le due parole da confrontare devono cominciare con la stessa lettera. Vedere il Capitolo 29 per ottenere dettagli riguardanti la modalità di ricerca di SOUNDEX.

ESEMPIO select Cognome

```

        from INDIRIZZO
        where SOUNDEX(Cognome) = SOUNDEX('SEPP');

```

| COGNOME | NOME | TELEFONO |
|---------|---------|--------------|
| SZEP | FELICIA | 214-522-8383 |
| SEP | FELICIA | 214-522-8383 |

SPACE (SQL*PLUS)

Vedere SET

SPAZIO LIBERO

Vedere ESTENSIONI LIBERE

SPOOL

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE SET, Capitoli 5 e 20

SINTASSI SPO[OL] [*file*]OFF|OUT];

DESCRIZIONE SPOOL comincia o interrompe il reindirizzamento dell'output di SQL*PLUS su un file host o sulla stampante. SPOOL *file* reindirizza l'output di SQL*PLUS al file indicato. Qualora non venga specificato il tipo di file, SPOOL ne aggiunge uno di default in maniera analoga a quanto fa SAVE. OFF interrompe il reindirizzamento, OUT interrompe il processo e manda il file in stampa. Per reindirizzare l'output senza visualizzarlo, si imposti SET TERMOUT OFF nel file di avvio prima delle istruzioni SQL interessate ma dopo il comando SPOOL. SPOOL senza parametri mostra il nome dell'attuale (o del più recente) file di spool.

SQL

SQL è il linguaggio ANSI standard utilizzato per manipolare le informazioni di un database relazionale. Viene impiegato dai sistemi di gestione di ORACLE e IBM DB2. SQL si pronuncia "squel", anche se è ormai accettata anche la sigla "S.Q.L.".

Lo standard ANSI X3.135-1989 definisce il linguaggio SQL standard, compresa l'opzione di "Integrity Enhancement Option" che comprende tutti i controlli di integrità di CREATE TABLE (*vedere VINCOLO DI INTEGRITÀ*). Lo standard ANSI X3-168-1989 definisce invece l'embedded SQL. Lo standard ansi ANSI X3.135-1992 noto anche come SQL2, definisce la generazione successiva di SQL e include molte nuove e interessanti caratteristiche. Combina il SQL standard e quello embedded. Lo standard SQL3 comprende una versione di PL/SQL, procedure e trigger di default e tipi di dati astratti.

SQL*LOADER

SQL*LOADER è un'utilità di caricamento dei dati all'interno di un database ORACLE. *Vedere SQLLOAD*.

SQL*NET

SQL*NET è un prodotto opzionale che, funzionando insieme all'RDBMS di ORACLE, permette a due o più computer di scambiare dati su una rete. L'ultima versione si chiama Net8. *Vedere l'Oracle Net8 User's Guide* e il Capitolo 21.

SQL*PLUS

Vedere SQLPLUS.

SQLCASE (SQL*PLUS)

Vedere SET.

SQLCODE

TIPO Funzione di errore PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE ECCEZIONI, SQLERRM

SINTASSI *variabile* := SQLCODE

DESCRIZIONE Questa funzione restituisce il codice dell'ultimo errore, ma al di fuori delle routine di gestione degli errori (*vedere* ECCEZIONI e WHEN) restituisce sempre 0.

SQLCODE non può essere utilizzato come parte di un'istruzione SQL (come nell'inserimento del codice di errore in una tabella di rapporti). È però possibile copiare il valore in una variabile e utilizzare quest'ultima:

```
sql_error := sqlcode;
insert into PROBLEMLOG values (sql_error);
```

SQLCONTINUE (SQL*PLUS)

Vedere SET.

SQLERRM

TIPO Istruzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE ECCEZIONE, EXCEPTION_INIT, PRAGMA, SQLCODE

SINTASSI SQLERRM[(*intero*)]

DESCRIZIONE Se non viene indicato alcun *intero* SQLERRM restituisce il messaggio d'errore relativo al SQLCODE corrente. Se invece viene fornito un intero viene riportato il messaggio d'errore il cui codice è il valore dell'intero. Anche questa funzione, come SQLCODE, non può essere utilizzata direttamente in una istruzione SQL e può essere utile premettere una assegnazione:

```
error_message := sqlerrm;
```

o per ottenere il messaggio d'errore associato al codice 1403:

```
error_message := sqlerrm(1403);
```

Al di fuori del gestore di eccezioni (*vedere* SQLCODE), questa funzione senza argomento restituisce sempre "normal, successful completion". Solo all'interno del gestore di eccezioni si ottengono i messaggi seguenti.

- Il messaggio associato a un codice d'errore di ORACLE.
- Le parole "User-defined exception" per le eccezioni scatenate dall'utente cui non si sia assegnato un messaggio.
- Un messaggio definito dall'utente caricato attraverso la PRAGMA-direttiva EXCEPTION_INIT.

SQLLOAD

TIPO Comando SQL*LOADER

PRODOTTI SQL*LOADER

VEDERE ANCHE *Oracle Server Utilities User's Guide, Installation and User's Guide*
SINTASSI SQLLOAD [USERID=*utente/password*] CONTROL=*file* [LOG=*file*]

```
[BAD=file]
DATA=file [DISCARD=file] [DISCARDMAX=intero] [SKIP=intero]
[LOAD=intero] [ERRORS=intero] [ROWS=intero]
[BINDSIZE=intero]
[SILENT=(ALL | [HEADER | FEEDBACK | ERROR | DISCARDS] )]
[DIRECT=TRUE]
```

DESCRIZIONE SQLLOAD carica in un database ORACLE i dati contenuti all'interno di file esterni. SQL*LOADER richiede due tipi di file: i file di dati che contengono le informazioni da caricare in ORACLE e i file di controllo che contengono le informazioni riguardanti il formato dei dati, i record e i campi contenuti nei file di dati, l'ordine in cui devono essere caricati e il nome dei file di dati. Le informazioni del file di controllo possono comunque essere conglobate nei file di dati.

SQLLOAD crea automaticamente un file di log (.log), un file con il resoconto dei problemi (.bad) e, se viene utilizzato DISCARDMAX un file di annullamento (.dsc). Si possono separare gli argomenti di SQLLOAD con delle virgolette. Possono essere introdotti assieme alle parole chiave (come USERID o LOG), o senza, ma in questo caso devono essere riportati nell'ordine in cui sono stati riportati nel FORMATO. Questo metodo, comunque, può confondere eventuali altri operatori che siano costretti a leggere il codice e anche lo stesso utente, dopo un certo intervallo di tempo.

Per chiarezza è bene abituarsi a inserire questo comando con le parole chiave, che sono sempre seguite da un segno di uguale (=) e dall'argomento appropriato.

Se USERID viene omesso ORACLE lo richiederà in seguito. Se dopo il segno di uguale è presente una barra obliqua (/) vengono utilizzati il nome utente OPS\$ ID e la password di default. Si può inserire una stringa di specificazione di un database SQL*NET per collegarsi a un database remoto e caricarvi i dati.

CONTROL specifica il nome del file contenente il formato dei dati e le istruzioni di caricamento. Se lo si omette ORACLE lo richiede in seguito. Se non si fornisce alcuna estensione ORACLE suppone che sia quella di default (ovvero .ctl).

LOG specifica il nome del file che conterrà il report di SQL*LOADER: il nome dei file impiegati, il numero complessivo di record trasferiti e così via. Se non si specifica alcun nome, SQLLOAD crea un file con lo stesso nome di quello di controllo ma con estensione .log.

BAD specifica il nome del file che conterrà l'elenco dei record che SQL*LOADER non ha potuto caricare. Se non si specifica alcun nome, SQLLOAD crea un file con lo stesso nome di quello di controllo ma con estensione .bad.

DATA specifica il nome del file che contiene i record da caricare. Se non si specifica alcun nome, SQLLOAD assume che abbia lo stesso nome del file di controllo ed estensione .dat. I dati possono essere inclusi anche nello stesso file usato come file di controllo.

DISCARD specifica il nome del file che contiene quei record che SQL*LOADER non ha né rifiutato né inserito nel database perché non soddisfacevano ai criteri della clausola when del file di controllo o perché contenevano solo campi vuoti. Se lo si indica questo file viene comunque creato (ma se non viene riscontrato nulla di anomalo alla fine sarà vuoto). Esso viene creato automaticamente solo nel caso in cui si specifichi DISCARDMAX nel file di controllo.

In quest'ultimo caso, se non si specifica alcun nome, SQLLOAD ne crea uno con lo stesso nome del file di controllo ma con estensione .dsc. I suoi record hanno lo stesso formato di quelli del file di dati e possono quindi essere facilmente caricati in seguito (se necessario).

DISCARDMAX specifica il numero massimo di record anomali prima di interrompere il caricamento. **DISCARDMAX=0** non consente di tralasciare alcun record. Eliminare del tutto la parola chiave significa consentire un numero qualsiasi di record anomali.

SKIP imposta il numero di file da saltare dall'inizio del file prima di cominciare il caricamento. Se la si omette non ne viene saltato nessuno.

LOAD imposta il numero massimo di record da caricare. Questo numero non comprende quelli eventualmente saltati. Se si omette questa parola chiave i record vengono caricati tutti.

ERRORS specifica il numero massimo di errori di inserimento prima che il caricamento venga interrotto. Per interrompere il trasferimento al primo errore si imposta **ERRORS=0**. Omettendo la parola chiave viene utilizzato il valore di default (50). Per consentire un numero illimitato di errori si specifichi un valore maggiore del numero dei record da caricare.

ROWS specifica il numero di righe da bufferizzare nei comandi di inserimento e salvataggio. Il valore di default è 64 e viene utilizzato quando viene omessa la parola chiave.

BINDSIZE imposta il massimo numero di byte da utilizzare per la bufferizzazione delle righe nei comandi di inserimento e salvataggio discussi in **ROWS**. **BINDSIZE** sovrascrive il valore di default (detto Bind Array) del computer in uso, che varia da macchina a macchina. Se il valore fornito è minore del numero di byte richiesto da **ROWS**, **BINDSIZE** riduce automaticamente il numero di righe bufferizzabili.

SILENT impone a SQLLOAD di eliminare alcuni messaggi informativi.

- **HEADER** elimina l'intestazione di SQL*LOADER.
- **FEEDBACK** elimina il messaggio di risposta (feedback) di ciascun salvataggio.
- **ERRORS** elimina la registrazione (nel file di log) dei record che hanno causato un errore in ORACLE sebbene il loro numero venga comunque aggiornato.
- **DISCARDS** elimina la registrazione (nel file di log) dei record tralasciati, sebbene il loro numero venga comunque aggiornato.
- **ALL** elimina tutti i messaggi precedenti.

Se è necessario inserire più di un'opzione, occorre separarle con delle virgole e racchiuderle tra parentesi.

DIRECT richiama l'opzione di caricamento a percorso diretto (vedere oltre).

OPTIONS Oltre che sulla riga di comando, le ultime sei opzioni possono essere specificate anche nel file di controllo. Esse devono essere inserite all'inizio del file di seguito alla parola chiave **OPTIONS** con il formato seguente:

```
OPTIONS {[SKIP=intero] [LOAD=intero] [ERRORS=intero]
[ROWS=intero]
[BINDSIZE=intero] [SILENT=(ALL | [FEEDBACK | ERROR | DISCARDS] )]}
```

Si noti che l'opzione HEADER di SILENT non può essere utilizzata all'interno del file di controllo in quanto nel momento in cui SQLLOAD dovesse leggerla avrebbe comunque già visualizzato la sua intestazione. Per evitare la visualizzazione dell'intestazione di SQLLOAD non c'è alcuna alternativa alla riga di comando.

Il file di controllo Il file di controllo contiene diversi tipi di informazioni essenziali al caricamento dei dati. Se è presente la parola chiave **INFILE**, essa specifica il nome del file di dati. Se è presente **BEGINDATA**, invece, i dati devono essere caricati dal file di controllo stesso e si trovano immediatamente dopo questa parola chiave (cioè questa dev'essere l'ultimo comando del file di controllo). Se non sono presenti né **INFILE** né **BEGINDATA**, ORACLE suppone che i dati siano contenuti nel file specificato sulla riga di comando oppure in un file con lo stesso nome di quello di controllo ma con estensione **.dat**.

Il file di controllo può specificare altresì (anche se con una sintassi leggermente modificata) informazioni riguardanti i file **BAD** e **DISCARD**, i nomi delle tabelle da caricare, il formato dei record all'interno delle tabelle e le corrispondenze tra i campi nei record e nelle colonne. Può anche discriminare quali record caricare (mediante **when**) e se i record debbano essere aggiunti al fondo della tabella, debbano sostituire quelli eventualmente presenti o se debbano essere inseriti all'interno di una tabella vuota.

Nel file di controllo è permesso spezzare le istruzioni su righe diverse. Si possono utilizzare parole chiave di SQLLOAD come nomi di colonna o di tabella racchiudendoli tra doppi apici. Anche se SQLLOAD non fa differenza fra maiuscole e minuscole, i doppi apici rendono il loro contenuto un letterale, quindi ci si premuri di scrivere i nomi in maiuscolo.

Nei file di controllo di SQLLOAD i commenti cominciano con **--** (due trattini). SQLLOAD ignora qualsiasi cosa a partire da **--** fino alla fine della riga. Il loro uso non è consentito all'interno dei file di dati o nella parte del file di controllo dedicata ai dati (quella che segue **BEGINDATA**).

I comandi all'interno del file di controllo ridefiniscono eventuali opzioni imparate sulla riga di comando.

Formato del file di controllo

```

OPTIONS ({[SKIP=intero] [LOAD=intero]
[ERRORS=intero][ROWS=intero]
[BINDSIZE=intero] [SILENT=(ALL | [FEEDBACK | ERROR | DISCARDS] )}

LOAD [DATA]
[ { INFILE | INDDN} { file | * }
  [ STREAM | RECORD | FIXED lunghezza [BLOCKSIZE dimensione] |
  VARIABLE[lunghezza] ]
  [ { BADFILE | BADDN } file ]
  [ { DISCARDS | DISCARDMAX } intero ]

[ { INDDN | INFILE } ...]

[ APPEND | REPLACE | INSERT ]
[ RECLEN intero ]

```

```

[ { CONCATENATE intero |
  CONTINUEIF { [THIS | NEXT] (inizio[ :fine]) | LAST}
  operatore {'stringa' | X'hex'} } ]

INTO TABLE [utente.]tabella
  [APPEND | REPLACE | INSERT ]
  [WHEN condizione [ AND condizione ]...]
  [FIELDS [delimitatore] ]
(
  colonna      {
    RECNUM |
    CONSTANT valore |
    SEQUENCE ( { intero | MAX | COUNT } [ ,incremento ]) |
    [ POSITION ( { inizio [ fine } | * [+intero] } ) ]
tipo_dati
    [TERMINATED [BY] {WHITESPACE | [X]'character'}]
    [[OPTIONALLY] ENCLOSED [BY] [X]'character']
     [ NULLIF condizione ]
     [ DEFAULTIF condizione ]
    }
  [ ,... ]
)
[ INTO TABLE ... ]

[ BEGINDATA ]

```

Specifiche dei campi di dati FILE e INDDN sono sinonimi; a entrambi deve seguire il nome di un file contenente i dati da caricare. Se al posto di un nome di file viene inserito un * i dati sono da cercare nel file di controllo stesso, immediatamente dopo la parola chiave BEGINDATA (che richiede il metodo di lettura di default RECORD descritto più avanti). È possibile indicare diversi file di dati semplicemente ripetendo la parola chiave FILE o INDDN seguita dai nomi di file. Ogni file può avere associate diverse opzioni:

STREAM significa che i dati devono essere letti un byte per volta. I caratteri di avanzamento riga rappresentano i terminatori dei record fisici (i record logici possono essere costituiti di più record fisici, vedere CONCATENATE nel paragrafo “Costruzione di record logici” più avanti).

RECORD utilizza i metodi di gestione di file e record del sistema operativo in uso. Se i dati sono all’interno del file di controllo viene utilizzato questo metodo.

FIXED *lunghezza* significa che i record da leggere sono lunghi esattamente quel numero di byte, indipendentemente dal fatto che siano o meno disposti su righe diverse del file.

Se *lunghezza* vale 300, SQLLOAD legge 300 byte di dati sia che appaiano sotto forma di un unico record da 300 byte o 300 record da un solo byte. BLOCKSIZE specifica la dimensione dei blocchi nel caso questi debbano essere letti da nastro.

VARIABLE significa che i record verranno letti assumendo che i primi due byte di ciascuno ne contengano la lunghezza (questi byte non vengono ovviamente utilizzati per i dati). La lunghezza opzionale fornisce la massima dimensione dei record; se non viene specificata, ORACLE assume che sia di 8 Kbyte. Questo potrebbe supe-

rare il quantitativo di memoria disponibile su alcune macchine. Se si è certi che il valore sia inferiore a 8 K, specificandone uno inferiore è possibile migliorare sensibilmente le prestazioni.

Alcune delle opzioni descritte di seguito potrebbero non essere disponibili su alcuni sistemi.

BADFILE e BADDN sono sinonimi; a entrambi deve seguire il nome di un file in cui verranno scritti i dati rifiutati (quelli che ORACLE non ha potuto caricare nel database). Le regole di nomenclatura sono le stesse delle opzioni sulla riga di comando.

DISCARDFILE e DISCARDNN sono sinonimi; a entrambi deve seguire il nome di un file su cui verranno scritti i record non corrispondenti alla clausola when. Le regole di nomenclatura sono le stesse delle opzioni sulla riga di comando.

DISCARDS e DISCARDMAX sono sinonimi; il numero intero specifica il massimo numero di record anomali prima che SQLLOAD termini il caricamento del file corrente e passi al successivo (o termini se non ce ne sono altri). Le regole di nomenclatura sono le stesse delle opzioni sulla riga di comando. Il valore introdotto rimane lo stesso anche per i file successivi finché un'opzione analoga non lo reimposti.

Specifiche del metodo di caricamento di default per le tabelle Se le opzioni seguenti precedono la clausola into table, specificano il metodo di caricamento di tutte le tabelle che non lo specifichino da sole. L'utente deve avere il privilegio INSERT sulle tabelle da aggiornare.

APPEND aggiunge le righe al fondo della tabella.

INSERT inserisce le righe solo all'interno di una tabella vuota, in caso contrario annulla l'operazione.

REPLACE svuota la tabella e reinserisce le righe. L'utente deve avere il privilegio DELETE sulla tabella da aggiornare per poter utilizzare questa opzione.

Esistono tre opzioni equivalenti alle clausole resume yes, resume no e resume no replace di DB2. Per dettagli inerenti la compatibilità con DB2 vedere l'*Oracle Server Utilities User's Guide*.

Lunghezza massima dei record RECLEN viene utilizzato essenzialmente in due casi: quando SQLLOAD non è in grado di calcolare la lunghezza dei record automaticamente o quando si vuol trascrivere un intero record bad nel file bad. Nel secondo caso ORACLE scriverebbe normalmente solo la parte del record precedente l'errore. Se si vuole vederlo interamente, si specifichi manualmente la lunghezza: in questo modo i record la cui dimensione è inferiore alla lunghezza assegnata vengono trascritti per intero.

SQLLOAD calcola la lunghezza di default per i record a lunghezza variabile e assume che sia di 80 caratteri (salvo indicazioni contrarie) per i file a lunghezza fissa.

Costruzione di record logici Se è necessario comporre più record fisici in un unico record logico (ad esempio il Nome sulla prima riga, l'indirizzo sulla seconda, la Citta e il CAP sulla terza e così via), con CONCATENATE è possibile specificare l'intero che rappresenta il numero di record fisici da assemblare nel record logico (una tecnica di concatenazione alternativa combina i record fisici solo quando soddisfano alcuni requisiti. Questi vengono specificati dopo CONTINUEIF).

THIS verifica una condizione sul record corrente. Se è vera il record corrente viene concatenato.

NEXT verifica una condizione sul record successivo. Se è vera il record successivo viene concatenato a quello corrente.

inizio: *fine* o *inizio-fine* specificano le colonne su cui controllare l'esistenza della stringa di continuazione e quindi per decidere se effettuare la concatenazione. Il controllo viene effettuato tramite un operatore che può essere "uguale" o "non uguale". Uguale è sempre scritto = mentre non uguale può essere scritto !=, ~= oppure <>.

Per specificare ad esempio che i primi tre caratteri della stringa NEXT devono contenere 'WAG' perché vengano concatenati alla stringa corrente si può scrivere:

```
continueif next (1-3) = 'WAG'
```

In alternativa alla stringa di caratteri si può specificare una sequenza esadecimale preceduta dalla lettera X. Ad esempio:

```
continueif next (1-3) = X'0D031F'
```

verifica la presenza dei valori esadecimali 0D, 03 e 1F nelle posizioni dalla 1 alla 3.

I caratteri trovati in questo modo sono assunti come non facenti parte del record logico. Se il campo che denota la concatenazione deve essere conglobato nel record logico viene utilizzata la parola chiave last:

```
continueif last = ','
```

Tabelle da caricare INTO TABLE è seguito dal nome della tabella da caricare e, optionalmente, preceduto dal nome dell'utente se non si è i proprietari. È necessario disporre del privilegio INSERT sulla tabella. Se la tabella ha lo stesso nome di una parola chiave SQLLOAD, lo si racchiuda tra doppi apici e lo si digiti in maiuscolo:

```
into table Talbot."REGISTRO"
```

Per caricare diverse tabelle, si crei una sezione INTO TABLE per ciascuna di esse e le si metta nello stesso ordine in cui compaiono nel file di dati. Per caricare la stessa tabella da due file è necessario specificare due volte INTO TABLE. Per caricare due tabelle dallo stesso file occorre specificare quest'ultimo due volte.

WHEN è equivalente alla clausola where dell'istruzione select. Verifica delle condizioni sui record per decidere quali caricare nella tabella. WHEN può controllare sia il campo da caricare, sulla base della sua posizione all'interno del record logico, sia il valore da caricare nella colonna, attraverso il nome del campo. Se uno dei controlli fallisce non viene caricata nessuna riga. La condizione verificata è simile a quella utilizzata con CONTINUEIF (e anche con NULLIF e DEFAULTIF, esaminate tra breve). Il formato della condizione è il seguente:

```
inizio[ :fine] | colonna} operatore {'stringa' | X'hex'}
```

operatore può essere solo un'uguaglianza o una disuguaglianza (vedere CONTINUEIF), e diverse condizioni possono essere fuse tramite AND (OR non è disponibile). Ad esempio:

```
when (3-5) = 'SSM' and (22) '*'
```

oppure:

when Sezione = 'A'

Terminazione e inserimento di campi in un record FIELDS specifica i delimitatori che eventualmente separano i campi nei record. Il formato di FIELDS è il seguente:

```
FIELDS [TERMINATED [BY] {WHITESPACE | [X]'character'}]
  [[OPTIONALLY] ENCLOSED [BY] [X]'character']]
```

TERMINATED legge i campi a partire dalla fine del campo precedente fino al carattere delimitatore, il quale non viene considerato come parte dei dati. WHITESPACE significa che il terminatore è una sorta di spazio bianco (cioè uno spazio, un carattere di tabulazione, un avanzamento riga o pagina e così via). In alternativa può essere indicato un carattere tra apici o, premettendo una X, un numero esadecimale come X'1B'.

OPTIONALLY ENCLOSED indica che i dati possono essere terminati anche dal carattere TERMINATED (ad esempio un record in cui ciascun campo è separato da virgole). L'uso di OPTIONAL richiede necessariamente TERMINATED. Tutte le opzioni relative ai campi (eccetto la parola FIELDS) possono essere utilizzate anche per le colonne. Se in questo caso viene utilizzato OPTIONAL, TERMINATED dev'essere qui o, al massimo, nell'istruzione FIELDS.

ENCLOSED significa che i dati si trovano tra due delimitatori. Se vengono utilizzati sia ENCLOSED che TERMINATED, il loro ordine stabilisce l'ordine di valutazione.

BY è puramente opzionale e serve semplicemente a migliorare la leggibilità.

Definizione delle colonne *colonna* è il nome di una colonna della tabella. La specifica di ogni colonna è separata per mezzo di virgole e l'elenco è racchiuso tra parentesi. Ogni colonna può ottenere il proprio valore in quattro modi diversi:

RENUM significa che a questa colonna verrà assegnato un valore intero (sequenziale) uguale al numero del record logico letto dal file di dati.

CONSTANT significa che a questa colonna verrà assegnato il valore costante che segue la parola chiave CONSTANT. Nel caso di colonne CHAR è necessario che la costante sia racchiusa tra apici.

SEQUENCE differisce da RE NUM per il fatto che la sequenza può partire da un numero scelto arbitrariamente e incrementato a passi maggiori di uno. Ha il formato seguente:

```
SEQUENCE ( {intero | MAX | COUNT} [,incremento]
```

La sequenza può iniziare in uno dei tre modi seguenti: da *intero*, da MAX (cioè dal massimo valore attualmente archiviato nella colonna + incremento), da COUNT (cioè dal numero di righe già presenti nella tabella + incremento). *incremento* vale 1 per default ma può valere un numero qualsiasi da 1 in su.

I numeri di sequenza vengono incrementati ogni volta che viene aggiunta o respinta una riga. In questo modo però i record respinti producono indici inesistenti; i valori mancanti devono essere inseriti in seguito e i loro numeri di sequenza devono essere assegnati manualmente.

POSITION specifica la posizione dei dati delle colonne dei record logici. Questa posizione può essere indicata in termini assoluti o relativi alla colonna precedente. Il formato di POSITION è il seguente:

```
POSITION ( {inizio[:fine] | * [+intero] } )
```

inizio è la posizione iniziale della colonna del record logico mentre *fine* segnala opzionalmente la posizione finale. Tra *inizio* e *fine* possono essere introdotti i due punti o il trattino. Se non viene specificata la posizione finale SQLLOAD assume una lunghezza di default per la colonna sulla base suo del tipo di dati. Le colonne di tipo CHAR hanno lunghezza di default pari a 1.

Un * significa che questo campo comincia immediatamente dopo il termine di quello precedente. +*intero* indica il numero di posizioni da saltare per posizionarsi sull'inizio del record successivo.

Se non viene utilizzato POSITION il metodo utilizzato per default è POSITION(*). Nel caso venga impiegato più di un metodo per determinare la posizione di un campo di dati (ad esempio delimitatori e posizioni) vengono seguite alcune regole di precedenza. L'elenco che segue mostra l'ordine di priorità in ordine decrescente.

- Lunghezza di un tipo di dati specifico.
- Posizioni *inizio* e *fine*.
- Delimitatori.
- Posizione di *inizio* da sola.

Definizione del tipo di dati Sono 14 i tipi di dati utilizzabili nel caricamento tramite SQLLOAD:

```
CHAR  
DATE  
DECIMAL EXTERNAL  
DECIMAL  
DOUBLE  
FLOAT  
FLOAT EXTERNAL  
GRAPHIC  
GRAPHIC EXTERNAL  
INTEGER  
INTEGER EXTERNAL  
SMALLINT  
VARCHAR  
VARGRAPHIC
```

Di seguito sono elencate le opzioni di formattazione per ciascuno di essi.

Per i tipi carattere

```
CHAR [(lunghezza)] [delimitatore]
```

Se non viene specificata alcuna *lunghezza* vengono utilizzati i valori di POSITION. Se neanche questi sono stati impostati la lunghezza viene impostata per default a 1. Se la colonna del database è LONG è indispensabile che qui o in POSITION

venga impostata una lunghezza. Vedere “Terminazione e inserimento dei campi nei record” per dettagli relativi all’uso del *delimitatore*.

Per i tipi data

DATE [(*lunghezza*)] ['*formato di data*'] [*delimitatore*]

Se non viene specificata alcuna *lunghezza* vengono utilizzati i valori di POSITION. Il formato di data può essere uno dei formati utilizzati da TO_DATE (vedere DATA, FORMATI). Se non viene specificato alcun formato, viene utilizzato il formato di default di ORACLE (DD-MON-YY). Vedere “Terminazione e inserimento di campi in un record” per dettagli relativi all’uso del *delimitatore*.

Per i numeri decimali in formato carattere

DECIMAL EXTERNAL [(*lunghezza*)] [*delimitatore*]

Questo formato viene utilizzato per i numeri decimali presenti sotto forma di caratteri piuttosto che in formato binario impaccato. Questi numeri vengono quindi trattati come caratteri e usano lo stesso formato di CHAR. Quando però si utilizza DEFAULTIF (esaminato tra breve), se si vuole impostare il valore di default a NULL si utilizza CHAR. Vedere “Terminazione e inserimento di campi in un record” per dettagli relativi all’uso del *delimitatore*.

Per i numeri decimali in formato compattato

DECIMAL (*cifre* [,*precisione*])

Nel formato decimale compattato a ogni cifra corrispondono 4 bit. È di 4 bit anche la rappresentazione del segno. Il parametro *cifre* indica il numero di cifre del numero (333.33 ne ha 5, non 6). La *precisione* indica quante cifre mantenere a destra del punto decimale (che è隐式的). La *precisione* può essere maggiore del numero di cifre, nel qual caso si ottiene un numero minore di uno con alcuni zeri dopo il punto. I due parametri devono essere positivi. Se non viene specificata alcuna precisione, si suppone che il numero sia intero.

Il numero di posizioni-carattere richiesto da un numero intero in base 10 nell’archiviazione su file è uguale a (*cifre*+2)/2.

Per i numeri binari a virgola mobile e doppia precisione

DOUBLE

In questo tipo di dati viene utilizzata la posizione di partenza della clausola position e la lunghezza viene ricavata dalla dimensione dei tipi analoghi sul computer in uso. Corrispondono ai tipi di dati DOUBLE e LONG FLOAT del linguaggio C.

Per i numeri binari a virgola mobile e singola precisione

FLOAT

In questo tipo di dati viene utilizzata la posizione di partenza della clausola position e la lunghezza viene ricavata dalla dimensione del tipo analogo sul computer in uso. Corrisponde al tipo FLOAT del linguaggio C.

Per i numeri in virgola mobile in formato carattere

FLOAT EXTERNAL [(*lunghezza*)] [*delimitatore*]

Il formato utilizzato per i numeri in virgola mobile (siano essi a singola o a doppia precisione) per la loro archiviazione su file è il formato carattere. Questi numeri sono quindi trattati come stringhe di caratteri e seguono lo stesso formato dei tipi CHAR. Quando però si utilizza DEFAULTIF (descritto tra breve), se si vuole che il valore di default sia NULL, si utilizzi CHAR. Se invece si vuole che sia 0, si utilizzi DECIMAL EXTERNAL. Se non viene specificata alcuna lunghezza, viene utilizzata quella impostata in POSITION. Vedere “Terminazione e inserimento di campi in un record” per dettagli relativi all’uso del *delimitatore*.

Per i dati provenienti da stringhe di caratteri a doppio byte (DBCS)

GRAPHIC [(*lunghezza*)]

Anche se ORACLE non supporta direttamente le stringhe di caratteri a doppio byte, è in grado di leggere i dati sotto forma di stringa a byte singolo e archiviarli in una colonna. La lunghezza può essere definita in due modi: specificandola nei primi due byte della stringa nel qual caso è uguale alla metà del numero di byte necessari per l’archiviazione. Una stringa con 12 byte da 8 bit avrà lunghezza 6; nei primi due byte viene scritto 06 (questi due byte non sono compresi nella lunghezza). La seconda definizione prevede che la lunghezza venga ricavata indirettamente tramite la loro posizione iniziale e finale nel file e viene misurata in byte singoli.

GRAPHIC EXTERNAL [(*lunghezza*)]

È identico a GRAPHIC senza EXTERNAL, salvo per il fatto che in questo caso si suppone che i campi siano racchiusi tra caratteri di delimitazione. La *lunghezza*, qualora venga specificata esplicitamente, non comprende i caratteri di delimitazione (shift-in e shift-out).

Per gli interi binari long

INTEGER

Per questo tipo di dati viene utilizzata solo la posizione di partenza indicata nella clausola position e la lunghezza viene ricavata da quella del tipo analogo sul sistema in uso. Il tipo corrispondente in C è LONG INT.

Per gli interi binari short

SMALLINT

Per questo tipo di dati viene utilizzata solo la posizione di partenza indicata nella clausola position e la lunghezza viene ricavata da quella del tipo analogo sul sistema in uso. Il tipo corrispondente in C è SHORT INT.

Per i numeri interi in formato carattere

INTEGER EXTERNAL [(*lunghezza*)] [*delimitatore*]

Questo formato viene utilizzato per i numeri interi rappresentati nei file mediante caratteri. Questi numeri seguono lo stesso formato dei tipi CHAR. Quando però si utilizza DEFAULTIF (esaminato tra breve), se si vuole che il valore di default sia NULL, si impieghi CHAR. Se si vuole che sia 0, si utilizzi DECIMAL EXTERNAL. Se non viene specificata alcuna lunghezza, vengono utilizzati i valori di position. Vedere “Terminazione e inserimento di campi in un record” per dettagli relativi all’uso del *delimitatore*.

Per le stringhe di caratteri a lunghezza variabile

VARCHAR [(lunghezza)]

Per questo tipo di dati viene utilizzata la posizione di partenza di position mentre la lunghezza viene ottenuta dai primi due byte della stringa. La *lunghezza* riportata qui è solo quella massima del campo e non comprende i due byte che specificano la lunghezza. Il suo scopo è semplicemente quello di favorire l’ottimizzazione dei buffer.

Per le stringhe di caratteri a doppio byte (DBCS)

VARGRAPHIC [(lunghezza)]

Anche se ORACLE non supporta direttamente le stringhe di caratteri a doppio byte, è in grado di leggere i dati sotto forma di stringa a byte singolo e archiviarli in una colonna. La lunghezza viene riportata nei primi due byte della stringa; la posizione di questi due byte viene specificata dalla parola chiave POSITION. La lunghezza è espressa in doppi byte. È la metà del numero di byte necessari per l’archiviazione. Una stringa con 12 byte da 8 bit ha lunghezza 6; nei primi due byte viene scritto 06 (questi due byte non sono compresi nella lunghezza).

Impostazione completa o condizionale di una colonna a NULL Una colonna può essere impostata a NULL in maniera condizionale utilizzando la parola chiave NULLIF:

NULLIF {(inizio[:fine]} | colonna} operatore {'string' | X'hex'}

operatore può essere un’uguaglianza o una disuguaglianza. Uguale si scrive sempre = mentre “non uguale” può scriversi !=, ~ o <>. Si possono congiungere condizioni diverse per mezzo di AND (OR non è disponibile). Ad esempio:

NULLIF (37-39) = 'DOG' and (22-25) = 'FRED'

oppure:

NULLIF Eta = '65'

Se si vuole che i valori di una colonna vengano impostati a NULL per ogni nuova riga inserita, basta non includere il nome della colonna all’interno del file di controllo.

Impostazione condizionale di una colonna numerica a zero La parola chiave DEFAULTIF può essere utilizzata in maniera analoga a NULLIF:

DEFAULTIF {(inizio[:fine]} | colonna} operatore {'string' | X'hex'}

Le due parole chiave sono identiche sulle colonne carattere. Nelle colonne numeriche però DEFAULTIF imposta i valori delle colonne a zero al verificarsi di alcune condizioni mentre NULLIF imposta i valori su NULL. In genere uno solo dei due è sufficiente ma possono esistere dei casi in cui alcune celle devono valere 0 e altre NULL sulla base di condizioni diverse. DEFAULTIF tratta DECIMAL EXTERNAL, FLOAT EXTERNAL e INTEGER EXTERNAL come numeri.

SQLNUMBER (SQL*PLUS)

Vedere SET.

SQLPLUS

TIPO Prodotto ORACLE

PRODOTTI SQL*PLUS

VEDERE ANCHE Capitoli 5, 13 e 20.

SINTASSI SQLPLUS [*utente[/password]*] [*@database*] [*@file*] [-SILENT] |
[*/NOLOG*] [-SILENT] | [-?]

DESCRIZIONE SQLPLUS avvia SQL*PLUS. Se si specificano solo il nome utente e la password SQL*PLUS si collega al database di default. Se si digita il solo nome utente SQL*PLUS richiede la password, la quale non viene mai visualizzata. Se si specifica *@database* si ottiene la connessione al database specificato. Il *@database* può essere situato ovunque fintanto che il computer locale sia collegato a una rete tramite SQL*NET. Non devono essere presenti spazi tra *password* e *@database*. Per ulteriori informazioni riguardanti SQL*NET, vedere la *SQL*Net User's Guide* e il Capitolo 21. Il *@file* viene utilizzato come file di avvio non appena caricato SQL*PLUS. Prima di *@file* dev'essere presente uno spazio. Se il nome utente e la password non vengono digitati sulla riga di comando di SQLPLUS devono presenziare sulla prima riga del file. In caso siano presenti sia nel file che nella riga di comando si ottiene un messaggio di errore che non influenzera comunque la corretta connessione al database. Per inserire il nome utente e la password nel file li si separi tramite una barra obliqua (/):

GEORGE/MISTY

/NOLOG lancia SQL*PLUS ma non effettua il collegamento a ORACLE. Per collegarvisi in seguito sarà necessario utilizzare il comando CONNECT (*vedere CONNECT*).

-SILENT sopprime tutto l'output di SQL*PLUS, compresi i prompt di comandi e persino le righe di logon e copyright. In questo modo è possibile che altri programmi usino SQL*PLUS in maniera trasparente.

-? visualizza la versione corrente di SQL*PLUS senza avviarlo.

ESEMPIO Un comando di avvio classico di SQL*PLUS:

```
sqlplus george/misty
```

Per collegarsi contestualmente al database EDMESTON si utilizza:

```
sqlplus george/misty@EDMESTON
```

Per avviare il file di report REPORT6 che contiene nome utente e password sulla prima riga si utilizza:

```
sqlplus @report6
```

Per avviare il file di report REPORT6 e contestualmente collegarsi al database EDMESTON si utilizza:

```
sqlplus george/misty@EDMESTON @report6
```

SQLPREFIX (SQL*PLUS)

Vedere SET.

SQLPROMPT (SQL*PLUS)

Vedere SET.

SQLTERMINATOR (SQL*PLUS)

Vedere SET.

SQRT

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE POWER

SINTASSI SQRT(*valore*)

DESCRIZIONE SQRT estrae la radice quadrata di *valore*.

ESEMPI SQRT(64) = 8

SQRT(d2) = 1.414...

SQRT(-1) produce un messaggio di errore.

La radice quadrata di un numero negativo non è ricavabile in ORACLE (matematicamente è un numero immaginario).

START

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE @@, ACCEPT, DEFINE, SPOOL

SINTASSI STA[RT] *file* [*parametro*] [*parametro*]...

DESCRIZIONE START esegue il file di avvio specificato il quale può contenere qualsiasi istruzione SQL*PLUS valido. Se non viene specificato alcun tipo di file, START assume che sia .sql. Gli eventuali parametri sulla riga di comando vengono inseriti all'interno di variabili di nome &1, &2, &3 e così via, e ricevono i parametri in ordine da sinistra a destra. I parametri possono contenere un solo numero o una sola parola.

ESEMPIO start skill HELEN

dove il file skill.sql contiene le righe seguenti:

```
select * from COMPITOLAVORATORE
where Nome LIKE '&1%';
```

Produce il risultato seguente:

| NOME | COMPITO | CAPACITA |
|--------------|-------------|--------------|
| HELEN BRANDT | TREBBIATORE | MOLTO VELOCE |

STDDEV

TIPO Funzione di gruppo SQL

PRODOTTI SQL*PLUS

VEDERE ANCHE FUNZIONI DI GRUPPO, VARIANCE, Capitolo 7

SINTASSI STDDEV(*valore*)

DESCRIZIONE STDDEV restituisce la deviazione standard statistica della distribuzione gaussiana normale di un gruppo di valori. Nel calcolo vengono ignorati eventuali valori NULL.

STORAGE

TIPO Clausola di un comando SQL

PRODOTTI Tutti

VEDERE ANCHE BLOCK, CREATE CLUSTER, CREATE INDEX, CREATE ROLLBACK SEGMENT, CREATE SNAPSHOT, CREATE SNAPSHOT LOG, CREATE TABLE, CREATE TABLESPACE e le analoghe istruzioni ALTER

SINTASSI STORAGE ([INITIAL *intero* [K|M]]

[NEXT *intero* [K|M]]

[PCTINCREASE *intero*]

[MINEXTENTS *intero*]

[MAXEXTENTS *intero*:UNLIMITED]

[OPTIMAL {*intero* [K|M] | NULL}]

[FREELISTS *intero*]

[FREELIST GROUPS *intero*])

DESCRIZIONE La clausola STORAGE può essere utilizzata in tutte le 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) di dimensione dipendente dal sistema operativo in uso.

INITIAL alloca ad un oggetto la prima parte dell’estensione libera. Se non viene specificata questa opzione si usano per default cinque blocchi. L’estensione iniziale più piccola che si possa allocare è di due blocchi, mentre la più grande dipende dal sistema operativo. È possibile esprimere queste dimensioni sotto forma di numeri interi o come interi seguiti da K o M per indicare rispettivamente kilobyte o megabyte.

NEXT rappresenta la dimensione dei blocchi da allocare quando viene riempita l’estensione iniziale. Se non viene specificato altro, l’impostazione di default è di cinque blocchi.

L’estensione iniziale più piccola che si possa allocare è di un blocco, mentre la più grande dipende dal sistema operativo. È possibile esprimere queste dimensioni sotto forma di numeri interi o come interi seguiti da K o M per indicare rispettivamente kilobyte o megabyte.

PCTINCREASE controlla il tasso di crescita per secondo delle estensioni. Se viene impostato a 0, qualsiasi estensione aggiuntiva avrà la stessa dimensione della seconda, specificata con NEXT. Se PCTINCREASE è un numero intero positivo, ogni estensione successiva sarà più grande della precedente della percentuale specificata. Ad esempio, se PCTINCREASE vale 50 (l’impostazione di default), ogni estensione aggiuntiva sarà del 50 per cento più grande della precedente. PCTINCREASE non può essere negativo. Il valore minimo è zero mentre il massimo di

pende dal sistema operativo. ORACLE arrotonda la dimensione delle estensioni al multiplo intero successivo di blocchi di sistema.

MINEXTENTS vale 1 per default (o 2 per i segmenti di rollback): quando viene creato l'oggetto viene allocata solo l'estensione iniziale. Un numero maggiore di 1 crea diverse estensioni (che non necessariamente devono essere contigue su disco) la dimensione delle quali viene determinata dai valori di INITIAL, NEXT e PCTINCREASE.

MAXEXTENTS imposta il numero massimo di estensioni allocabili. Il numero minimo è 1, il massimo dipende dal sistema operativo. A partire da ORACLE7.3 è possibile specificare anche **UNLIMITED** (illimitato).

OPTIMAL imposta la dimensione ottimale di un segmento di rollback espressa in byte. Si possono utilizzare K e M per i kilobyte e i megabyte. ORACLE dealloca dinamicamente le estensioni non richieste in modo da mantenere la dimensione ottimale. **NONE** significa che ORACLE non effettua alcuna disallocazione nei segmenti di rollback (ed è il comportamento di default).

FREELIST GROUPS imposta il numero di gruppi di free list. Il valore di default è 1. Questa impostazione è adottata nell'opzione Parallel Server di ORACLE sulle tabelle, sui cluster e sugli indici.

FREELISTS imposta il numero di free list in ogni gruppo.

STORE

PRODOTTI SQL*PLUS

VEDERE ANCHE SAVE, START, Capitolo 13

SINTASSI STORE SET *file[.ext]* [CRE[ATE] | REP[LACE] | APP[END]]

DESCRIZIONE STORE salva tutte le impostazioni attuali dell'ambiente SQLPLUS in un file.

ESEMPIO Il comando seguente salva l'ambiente SQLPLUS corrente in un file di nome SETTINGS.SQL.

```
store settings.sql
```

STRUCTURED QUERY LANGUAGE

Vedere SQL.

SUBSTR

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE ||, CARATTERI, FUNZIONI; INSTR, Capitolo 6

SINTASSI SUBSTR(*stringa*, *inizio* [,*conto*])

DESCRIZIONE SUBSTR taglia una *stringa* partendo dal carattere la cui posizione è *inizio* terminando dopo un numero di caratteri definito da *conto*. Se quest'ultimo parametro non viene specificato la stringa viene tagliata a partire da *inizio* verso la fine della stringa.

ESEMPIO SUBSTR('NEWSPAPER',5)

produce:

PAPER

SUBSTRB

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE `||`, CARATTERI, FUNZIONI; INSTRB, SUBSTR, Capitolo 6

SINTASSI `SUBSTRB(stringa, inizio [,conto])`

DESCRIZIONE `SUBSTRB` taglia una parte di *stringa* a partire dal byte *inizio* fino al byte *conto*. Se quest'ultimo parametro non viene specificato la stringa viene tagliata da sinistra a destra a partire da *inizio*. Questa funzione permette di tagliare stringhe di caratteri multibyte, un byte alla volta.

SUFFIX (SQL*PLUS)

Vedere SET.

SUM

TIPO Funzione di gruppo SQL

PRODOTTI Tutti

VEDERE ANCHE COMPUTE, FUNZIONI DI GRUPPO, Capitolo 7

SINTASSI `SUM([DISTINCT] valore)`

DESCRIZIONE `SUM` restituisce la somma di tutti i *valori* di un gruppo di righe. DISTINCT fa sì che la somma venga operata solo su valori distinti, opzione generalmente poco utile.

SVRMGRL

SVRMGRL richiama Server Manager in modalità a linea di comando. Server Manager viene utilizzato dal DBA durante la gestione e il monitoraggio del database.

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 ha il possesso della maggior parte delle tabelle del dizionario mentre SYSTEM possiede le viste create sulla base delle tabelle precedenti.

SYSDATE

TIPO Pseudocolonna SQL

PRODOTTI SQL

VEDERE ANCHE PSEUDOCOLONNE

SINTASSI `SYSDATE`

DESCRIZIONE `SYSDATE` contiene la data e ora correnti. Restituisce un tipo DATE.

SYSTEM (TABLESPACE)

SYSTEM è il nome del tablespace originale.

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 ha il possesso della maggior parte

delle tabelle del dizionario mentre SYSTEM possiede le viste create sulla base delle tabelle precedenti.

SYSTEM GLOBAL AREA (SGA)

La *System Global Area* è un'area di archiviazione posta nella memoria principale (o in quella virtuale, a seconda del sistema operativo in uso) che è alla base dell'attività di ORACLE durante la manutenzione del database. La dimensione della SGA e le prestazioni del sistema dipendono dai valori dei parametri di init.ora. La SGA consente la comunicazione tra utenti e processi in background.

TABELLA

La tabella è la struttura dati fondamentale per i sistemi di gestione dei database relazionali. Una tabella consiste di una o più unità d'informazione (riga) ciascuna contenente gli stessi tipi di valori (colonne). *Vedere CREATE TABLE*.

TABELLA DI SOLO INDICE

Una tabella di solo indice mantiene un ordinamento dei dati relativo ai valori della colonna primaria. In questo modo è possibile archiviare tutti i dati di una tabella all'interno di un indice.

Visto che i dati della tabella vengono archiviati all'interno di un indice, le righe della tabella non hanno RowID. Perciò non è possibile selezionare i valori delle pseudocolonne RowID di tabelle a indice unico. Non è inoltre possibile creare indici aggiuntivi; l'unico indice valido è quello della chiave primaria.

Per creare una tabella a indice unico si utilizzi l'opzione organization index del comando *create table* come si vede nell'esempio seguente:

```
create table TROUBLE (
    Citta          VARCHAR2(13),
    DataCampione   DATE,
    Mezzogiorno    NUMBER(4,1),
    Mezzanotte     NUMBER(4,1),
    Precipitazione NUMBER,
    constraint TROUBLE_PK PRIMARY KEY (Cita, DataCampione))
organization index;
```

Per far sì che TROUBLE sia una tabella a indice unico occorre generare un vincolo di chiave primaria.

Vedere il Capitolo 17 e CREATE TABLE.

TABELLA OGGETTO

Una tabella oggetto è una tabella in cui ogni riga è un oggetto riga. *Vedere il Capitolo 31 e la parte relativa al comando CREATE TABLE.*

TABELLA ANNIDATA

Le tabelle annidate sono strutture disponibili a partire da ORACLE8. Una tabella annidata è, come suggerisce il nome, una tabella contenuta all'interno di un'altra. In questo caso si tratta di tabelle rappresentate sotto forma di colonne di altre tabelle. È possibile avere diverse righe della tabella interna associate a ciascuna riga della tabella principale. Non c'è alcun limite al numero di oggetti per riga. *Vedere il Capitolo 26 per dettagli relativi a creazione, utilizzo e query di tabelle annidate.*

TABELLA PARTIZIONATA

Una tabella partizionata è una tabella le cui righe siano state partizionate su diverse tabelle più piccole. Se si utilizzano le utilità di partizionamento di ORACLE il sistema mantiene una vista sulla tabella partizionata. Vedere il Capitolo 17.

TABLE, PL/SQL

TIPO Tipo di dati PL/SQL

PRODOTTI Tutti

VEDERE ANCHE TIPI DI DATI, RECORD (PL/SQL)

SINTASSI TYPE *nuovo tipo* IS TABLE OF
 {*tipo* | *tabella.colonna%TYPE*} [NOT NULL]
 INDEX BY BINARY_INTEGER;

DESCRIZIONE Una dichiarazione TABLE crea un nuovo tipo che può essere utilizzato nella dichiarazione delle variabili. 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 sia un tipo standard PL/SQL (compresi i RECORD ma escluse altre tabelle), o può essere un riferimento a un tipo di colonna già utilizzato in un'altra tabella del database. I campi possono avere anche 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, scatta l'eccezione NO_DATA_FOUND.

ESEMPIO type SkillTable is table(skill COMPITOLAVORATORE.Compito%TYPE);
 SkillRecord MySkill;
 MySkill(1) := 'SCAVATORE';
 MySkill(5) := 'FABBRO';

TABLESPACE

Un tablespace è un file o un insieme di file utilizzato per archiviare dati di ORACLE. Un database ORACLE è costituito dal tablespace SYSTEM più eventualmente altri tablespace. Vedere il Capitolo 19.

TABLESPACE, BLOCCHI

Vedere DIZIONARIO, BLOCCO.

TAN

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE ACOS; ASIN; ATAN; ATAN2; COS; NUMERI, FUNZIONI; SIN; TANH; Capitolo 7

SINTASSI TAN(*valore*)

DESCRIZIONE TAN restituisce la tangente di un angolo il cui *valore* dev'essere espresso in radianti.

ESEMPIO select TAN(135*3.141593/180) Tan -- tangente di 135 gradi in radianti

```
      from DUAL;
TAN
-----
-1
```

TANH

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE ACOS; ASIN; ATAN; ATAN2; COS; NUMERI, FUNZIONI; SIN; TAN; Capitolo 7

SINTASSI $\text{TANH}(\text{valore})$

DESCRIZIONE TANH restituisce la tangente iperbolica dell'angolo *valore*.

TERMINALE ASINCRONO

Un terminale asincrono è un terminale che trasmette e riceve dati un carattere per volta (invece che un campo o una pagina video per volta). *Vedere TERMINALE SINCRONO.*

TERMINALE SINCRONO

Un terminale sincrono è un terminale che riceve e trasmette i dati un campo o una schermata alla volta (piuttosto che un carattere alla volta). È anche detto TERMINALE A BLOCCHI. *Vedere anche TERMINALE ASINCRONO.*

TERMINATORE

PL/SQL può essere utilizzato all'interno di linguaggi host utilizzando il precompilatore ORACLE. In tal caso i blocchi PL/SQL vengono trattati come singole istruzioni SQL, che vengono racchiuse tra le direttive EXEC SQL EXECUTE ed ENDEXEC. Il carattere che segue ENDEXEC indica il termine del codice PL/SQL. In C il carattere terminatore è il punto e virgola (;), in COBOL è il punto (.), in FORTRAN il carattere di avanzamento riga.

TERMOUT (SQL*PLUS)

Vedere SET.

TEMA

Un tema è una particolare tipologia di documento. Esso può o meno presenziare esplicitamente nel documento stesso. Ad esempio, un documento riguardante un mutuo può avere come tema la parola "banca" anche se questa non appare nel documento. *Vedere il Capitolo 30 per informazioni riguardanti il supporto ConText alle ricerche tematiche.*

TIME (SQL*PLUS)

Vedere SET.

TIMING (forma 1, SQL*PLUS)

Vedere SET.

TIMING (forma 2, SQL*PLUS)

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE CLEAR, SET

SINTASSI TIMI[NG] [START *area* | STOP | SHOW];

DESCRIZIONE TIMING tiene conto del tempo trascorso da START a STOP attraverso un nome di area. START apre un'area di temporizzazione e le assegna come titolo *area*. Il titolo dev'essere di un'unica parola. Possono esistere contemporaneamente più aree. L'area più recente rimane quella corrente finché non viene cancellata. In quel momento diventa corrente quella immediatamente precedente. Questo significa che le aree di temporizzazione sono annidate. L'area più recente mostrerà il valore minimo. L'intervallo di tempo totale di ogni area di temporizzazione è il suo tempo di rete più tutti quelli delle aree create dopo di lei.

SHOW visualizza l'area corrente e il tempo trascorso.

STOP visualizza l'area corrente e il tempo trascorso, cancella l'area e rende corrente l'area immediatamente precedente (se esiste). Vedere l'*SQL*Plus User's Guide and Reference* del proprio sistema operativo per ulteriori dettagli.

ESEMPIO Per creare un'area di temporizzazione di nome 1 si digit:

```
timing start 1
```

Per vedere il nome dell'area corrente e il tempo trascorso senza interrompere il cronometraggio si digit:

```
timing show
```

Per vedere il nome dell'area corrente e il tempo trascorso chiudendo il cronometraggio e rendendo attiva l'area precedente si digit:

```
timing stop
```

TO_BINARY_INTEGER

TIPO Funzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE TABLE - PL/SQL

SINTASSI TO_BINARY_INTEGER(*stringa*)
TO_BINARY_INTEGER(*numero*)

DESCRIZIONE TO_BINARY_INTEGER converte una stringa di CHAR di VARCHAR2 o un NUMBER in un intero binario. È possibile utilizzarlo come indice di una tabella PL/SQL. La *stringa* deve contenere un numero valido.

TO_CHAR (forma 1, conversione di date e numeri)

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE DATA, FORMATI; DATA, FUNZIONI; NUMERI, FORMATI, TO_DATE, Capitolo 8

SINTASSI TO_CHAR(*data*, 'formato')
TO_CHAR(*numero*, 'formato')

DESCRIZIONE TO_CHAR riformatta un numero o una data sulla base del formato specificato. *data* dev'essere una colonna definita con il tipo DATE di ORACLE. Non può essere una stringa, neppure se è nel formato standard (GG-MMM-AA). L'unico modo di utilizzare una stringa è di farla passare attraverso la funzione TO_DATE. I formati possibili di questa funzione sono molteplici e sono elencati nelle tabelle di FORMATI NUMERICI e FORMATI DI DATA.

ESEMPIO Si noti il formato dell'esempio seguente. È necessario operare in questo modo per evitare che SQL*PLUS utilizzi il formato di data di TO_CHAR che è lungo circa 100 caratteri:

```
column Formatted format a30 word_wrapped heading 'Formatted'

select DataNascita, TO_CHAR(DataNascita,DD/MM/YY) Formattata
  from COMPLEANNO
 where Nome = 'VICTORIA';

DATANASCI Formattata
-----
20-MAY-49 20/05/49
```

TO_CHAR (forma 2, conversione di etichette)

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE TO_CHAR (forma 1), TO_LABEL

SINTASSI TO_CHAR(*etichetta*, 'formato')

DESCRIZIONE Gli utenti di Trusted ORACLE possono utilizzare la funzione TO_CHAR per convertire le etichette di tipo MLSLABEL nel tipo VARCHAR2 utilizzando un *formato* opzionale. Se il formato viene omesso, l'etichetta viene convertita in un tipo VARCHAR2 utilizzando il formato di default per le etichette. Vedere la *Trusted ORACLE Server Administrator's Guide*.

TO_DATE

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE DATA, FORMATI; DATA, FUNZIONI; TO_CHAR, Capitolo 8

SINTASSI TO_DATE(*stringa*, 'formato')

DESCRIZIONE TO_DATE converte la stringa il cui formato è quello specificato in una data ORACLE. Questa funzione accetta anche numeri 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 eccetto uno il loro formato deve corrispondere a quello descritto da *formato*. Nel solo caso che la stringa sia nel formato 'DD-MON-YY' il *formato* può essere omesso. Vedere DATA, FORMATI per un elenco di formati accettabili da TO_DATE.

ESEMPIO select TO_DATE('02/22/90','MM/DD/YY') from DUAL;

```
TO_DATE('
-----
22-FEB-90
```

TO_LABEL

TIPO Funzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE TIPI DI DATI, TO_CHAR (forma 2)

SINTASSI `TO_LABEL(stringa, formato)`

DESCRIZIONE `TO_LABEL` converte una stringa di CHAR o di VARCHAR2 in un'etichetta di sistema operativo sicuro di tipo RAW MLSLABEL utilizzando il *formato* di etichetta specificato.

TO_MULTI_BYTE

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE FUNZIONI DI CONVERSIONE, Capitolo 9

SINTASSI `TO_MULTI_BYTE(stringa)`

DESCRIZIONE `TO_MULTI_BYTE` converte la stringa di byte singoli in una stringa di caratteri multibyte. Se un carattere non ha un equivalente multibyte la funzione restituisce il carattere senza modifiche.

TO_NUMBER

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE FUNZIONI DI CONVERSIONE, Capitolo 9

SINTASSI `TO_NUMBER(stringa)`

DESCRIZIONE `TO_NUMBER` converte una stringa di caratteri in un formato 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 effettua conversioni di tipo automatiche, tranne che quando vengono utilizzate colonne di caratteri contenenti numeri nelle clausole order by o nelle comparazioni.

ESEMPIO `TO_NUMBER('333.46')`

TO_SINGLE_BYTE

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE FUNZIONI DI CONVERSIONE, Capitolo 9

SINTASSI `TO_SINGLE_BYTE(stringa)`

DESCRIZIONE `TO_SINGLE_BYTE` converte la stringa di caratteri multibyte nella sua rappresentazione a byte singoli utilizzando la tabella degli equivalenti. Se un carattere multibyte non ha alcun equivalente monobyte la funzione lo restituisce senza modificarlo.

TRANSAZIONE

Una transazione è un'insieme di istruzioni SQL che ORACLE tratta come un'unica entità. L'insieme di modifiche viene reso permanente tramite il comando COMMIT. Gli effetti di una transazione possono essere annullati in toto o in parte attraverso il comando ROLLBACK.

ORACLE gestisce le transazioni sia con meccanismi di blocco (*vedere LOCK*) che con un *modello di consistenza multiversione* che in sostanza si comporta come se ogni transazione disponesse di una propria copia del database, cioè crea diverse versioni concorrenti dello stesso database. Una transazione parte 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 consistenza di lettura dei dati a livello di istruzioni, cioè durante l'esecuzione di una query i dati rimangono congelati finché ORACLE non ha finito di inviarli al programma.

Se però una transazione ha diverse query, ciascuna è consistente con se stessa ma non è detto che lo siano l'una con l'altra. Se si vuole mantenere una visuale costante sui dati del database durante un'intera transazione, questa deve utilizzare un meccanismo di *coerenza a livello di transazione* che garantisce che una transazione non risenta degli effetti del salvataggio di altre transazioni concorrenti. Queste transazioni in sola lettura, riconoscibili perché iniziano con l'istruzione SET TRANSACTION READ ONLY, possono solo effettuare interrogazioni ed eseguire alcuni comandi di controllo (*vedere SET TRANSACTION*).

TRANSLATE

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE REPLACE, Capitolo 9

SINTASSI TRANSLATE(*stringa,if,then*)

DESCRIZIONE TRANSLATE controlla ogni carattere della *stringa* e verifica se è presente tra quelli specificati in *if*. Se c'è TRANSLATE annota la posizione in cui è stato trovato il carattere in *if* e guarda nella stessa posizione di *then*. Qualsiasi carattere trovi in questa posizione viene sostituito nella stringa originale.

ESEMPIO

```
select TRANSLATE('I LOVE MY OLD THESAURUS','AEIOUY','123456')
   from DUAL;
TRANSLATE('ILOVEMYOLDTHESAURUS')
-----
3 L4V2 M6 4LD TH2S15R5S
```

TRIGGER

Un trigger è una procedura 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. Vedere i Capitoli 23 e 25 per una discussione completa sui trigger e relativi esempi, e CREATE TRIGGER in questo glossario per la sintassi.

TRIMOUT (SQL*PLUS)

Vedere SET.

TRUNC (forma 1, date)

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE COLUMN; DATA, FUNZIONI; TRUNC (formato 2, numeri), Capitolo 8

SINTASSI TRUNC(*data*,*'formato'*)

DESCRIZIONE TRUNC restituisce il troncamento di una *data* sulla base di un *formato*. Se questo non viene specificato la data viene troncata alla mezzanotte del giorno in corso. Di seguito sono elencati i formati di troncamento possibili:

| FORMATO | SIGNIFICATO |
|--------------------------|---|
| cc, scc | secolo, tronca al primo gennaio del primo anno del secolo qualsiasi data fino alle 23:59:59 del 31 dicembre del secolo in corso. |
| syear,syyy.y.yy.yyy.yyyy | anno, tronca al primo gennaio qualsiasi data fino alle 23:59:59 del 31 dicembre. |
| q | quarto, tronca al primo giorno del quarto corrente qualsiasi data compresa in quel quarto fino alle 23:59:59 dell'ultimo giorno del quarto. |
| month,mon,mm | mese, tronca al primo del mese qualsiasi data fino alle 23:59:59 dell'ultimo del mese. |
| ww | tronca a lunedì qualsiasi data fino alle 23:59:59 di domenica. |
| w | tronca all'ultimo giorno della settimana uguale al primo del mese qualsiasi data compresa nei sette giorni precedenti (<i>vedere</i> la spiegazione di seguito). |
| ddd,dd,j | tronca alla mezzanotte del giorno corrente qualsiasi ora fino alle 23:59:59. È il formato di troncamento di default. |
| day,dy,d | tronca a domenica (primo giorno della settimana) qualsiasi data fino alle 23:59:59 di sabato. |
| hh,hh12, hh24 | tronca all'ultima ora intera qualsiasi numero di minuti fino a 59. |
| mi | tronca all'ultimo minuto intero qualsiasi numero di secondi fino a 59. |

ww produce la data del lunedì della settimana corrente impostando l'ora a mezzanotte per qualsiasi data della settimana fino a domenica notte alle 23:59:59. Qualsiasi ora di lunedì viene troncata alla mezzanotte.

w funziona in maniera analoga tranne che per il fatto che produce la data di quel giorno della settimana uguale a quello del primo del mese in corso. Se Ad esempio il primo del mese era un venerdì, qualsiasi data fino al martedì precedente alle 23:59:59 viene troncata a venerdì a mezzanotte. Qualsiasi ora di venerdì viene troncata alla mezzanotte.

Il risultato di TRUNC è sempre una data con l'ora impostata a mezzanotte.

TRUNC (forma 2, numeri)

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE COLUMN; NUMERI, FUNZIONI; TRUNC (forma 1, date), Capitolo 7

SINTASSI TRUNC(*valore*,*precisione*)

DESCRIZIONE TRUNC restituisce *valore* troncato alla *precisione* desiderata.

ESEMPIO TRUNC(123.45,1) = 123.4

TRUNC(123.45,0) = 123
 TRUNC(123.45,-1) = 120
 TRUNC(123.45,-2) = 100

TRUNCATE

TIPO Istruzione SQL

PRODOTTI Tutti

VEDERE ANCHE CREATE CLUSTER, DELETE, DROP TABLE, TRIGGER, Capitolo 17

SINTASSI TRUNCATE {TABLE [utente.]tabella | CLUSTER [utente.]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 di hash, *vedere CREATE CLUSTER*). Se si indica l'opzione DROP STORAGE, TRUNCATE dealloca lo spazio relativo alle righe eliminate; l'opzione REUSE STORAGE, invece, mantiene lo spazio allocato in attesa di nuove righe della tabella. DROP STORAGE è l'impostazione di default.

Il comando TRUNCATE è più veloce di DELETE dato che non genera alcuna informazione di rollback, non utilizza alcun trigger di cancellazione (per cui dev'essere utilizzato con cautela) e non registra nulla su file. L'uso di TRUNCATE non invalida gli oggetti a seguito della cancellazione delle righe.

Non è possibile effettuare il rollback di un'istruzione TRUNCATE.

TTITLE

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE ACCEPT, BTITLE, DEFINE, PARAMETRI, REPFOOTER, REPHEADER, Capitolo 13

SINTASSI TTI[TLE] [opzione [testo\variabile]... | OFF | ON]

DESCRIZIONE TTITLE pone un titolo (che può anche occupare più di una riga) all'inizio di ogni pagina di resoconto. OFF e ON disabilitano e riattivano la visualizzazione del testo senza modificarne il contenuto. TTITLE da solo visualizza le impostazioni correnti del *testo* o della *variabile*.

testo è una stringa che rappresenta il titolo del rapporto mentre *variabile* è una variabile utente o di sistema come SQL.LNO (numero di linea corrente), SQL.PNO (numero di pagina corrente, SQL.RELEASE (numero di versione di ORACLE), SQL.SQLCODE (codice di errore corrente) o SQL.USER (nome utente).

SQL*PLUS utilizza title in una forma diversa se riconosce nella prima parola dopo title un'opzione valida tra quelle elencate di seguito.

COL[UMN] *n* salta direttamente alla posizione *n* a partire dal margine sinistro della linea corrente.

S[KIP] *n* stampa *n* linee vuote. Se non viene specificato alcun numero ne viene stampata una sola. Se *n* è 0 non viene stampata alcuna riga e la posizione di stampa corrente diventa il margine sinistro della linea.

TAB *n* salta avanti (o indietro se *n* è negativo) di *n* posizioni.

LE[FT], CE[NTER] e RI[GHT] allineano a destra, a sinistra o al centro i dati della riga corrente. Qualsiasi stringa o variabile posta dopo questi comandi viene

giustificata in gruppo fino alla fine del comando o fino a un successivo LEFT, CENTER, RIGHT o COLUMN. CENTER e RIGHT utilizzano il valore impostato da SET LINESIZE per determinare la lunghezza della linea.

FORMAT *stringa* specifica il formato di visualizzazione del testo o delle variabili seguenti e segue la stessa sintassi dell'opzione FORMAT del comando COLUMN; ad esempio, FORMAT A12 o FORMAT \$999,990.99. Ogni volta che compare FORMAT le impostazioni precedenti vengono ridefinite. Se non viene specificato alcun formato, viene utilizzato quello impostato da SET NUMFORMAT. Se neanche questo è stato preventivamente impostato viene utilizzato quello di default di SQL*PLUS.

Le date vengono stampate sulla base del formato di default a meno che 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.

TIPO

Vedere TIPO DI DATI ASTRATTO e CREATE TYPE.

TYPE (interno SQL)

TIPO Funzione PL/SQL

PRODOTTI PL/SQL

SINTASSI EXEC SQL TYPE *tipo* IS *tipo_dati*

DESCRIZIONE PL/SQL permette di assegnare un tipo di dati esterno a un tipo 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 di tipi di dati esterni vedere la *Programmer's Guide to the Oracle Precompilers*.

UID

TIPO Pseudocolonna SQL

PRODOTTI SQL

VEDERE ANCHE PSEUDOCOLONNE

SINTASSI UID

DESCRIZIONE L'UID è un numero unico che ORACLE assegna a ciascun utente del database corrente. Può essere utilizzato nelle istruzioni select ma, non essendo una vera e propria colonna, non può essere modificato dall'utente.

UNDEFINE

TIPO Comando SQL*PLUS

PRODOTTI SQL*PLUS

VEDERE ANCHE ACCEPT, DEFINE, PARAMETRI, Capitoli 13 e 15

SINTASSI UNDEF[INE] *variabile*

DESCRIZIONE UNDEFINE rimuove la definizione di una variabile utente definita tramite ACCEPT, DEFINE o come parametro del comando START. Si possono elimi-

nare le definizioni di diverse variabili con un solo comando. Il formato viene riportato nel listato seguente:

```
undefined variable1 variable2 ...
```

ESEMPIO Per eliminare la definizione della variabile di nome Total si digiti:

```
undefined Total
```

UNDERLINE (SQL*PLUS)

Vedere SET.

UNION

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE INTERSECT, MINUS, OPERATORI DI QUERY, Capitolo 11

SINTASSI

```
select...
      UNION [ALL]
      select...
```

DESCRIZIONE UNION combina due query restituendo tutte le righe distinte di entrambi i select oppure, qualora sia indicata l'opzione ALL tutte le righe senza controllarne l'unicità (e quindi comprendendo eventuali doppiioni). Il numero di colonne e i tipi di dati devono essere identici da una query all'altra mentre non è richiesto che lo siano i nomi delle colonne. *Vedere il Capitolo 11 per una discussione riguardante le importanti differenze tra INTERSECT, UNION e MINUS.*

UNIONE DI HASH

L'unione di hash è un metodo che permette di unire due tabella. Comparve per la prima volta nella versione 7.3 di ORACLE. Questo metodo utilizza algoritmi di hash (simili a quelli utilizzati per i cluster di hash) per valutare quali righe soddisfino i criteri di unione. Sebbene utilizzino algoritmi simili, non esiste alcuna relazione diretta tra le unioni di hash e i cluster di hash.

UNITÀ LOGICHE DI LAVORO

Vedere TRANSAZIONE.

UNITÀ DI LAVORO

In ORACLE, una transazione equivale ad un'unità di lavoro logica e comprende tutte le istruzioni 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.

UNRECOVERABLE

Nelle versioni precedenti di ORACLE (fino a ORACLE7.2), la parola chiave UNRECOVERABLE veniva utilizzata per disabilitare la scrittura di rapporti di ripristino durante l'esecuzione di comandi quali CREATE TABLE AS SELECT o CREATE INDEX. Con ORACLE8 questa funzione è stata racchiusa nella clausola NOLOGGING. *Vedere CREATE TABLE e CREATE INDEX.*

UPDATE (forma 1, interno SQL)

TIPO Istruzione SQL interna

PRODOTTI Precompilatori

VEDERE ANCHE EXECUTE IMMEDIATE, FOR, PREPARE, SELECT (forma 2)

SINTASSI EXEC SQL [AT database | :variabile] [FOR :intero]
 UPDATE [utente.]tabella[@dblink] [alias]
 SET { colonna = espressione [,colonna = espressione]... |
 (colonna [,colonna]...) = (sottoquery) }
 [WHERE condizione | CURRENT OF cursore];

DESCRIZIONE Vedere la descrizione delle varie clausole nel formato 3 di UPDATE. Gli elementi tipici dell'SQL interno sono i seguenti:

- AT database, che opzionalmente contiene il nome di un database precedentemente aperto con CONNECT.
- FOR :intero, che imposta il numero massimo di righe da prelevare. *intero* è il nome di una variabile host.
- *espressione* può comprendere una la variabile host *:variabile[:indicatore]*.
- La clausola where può contenere variabili e array host.
- CURRENT OF aggiorna la riga puntata da cursor. Il cursore dev'essere aperto e posizionato sulla riga. In caso contrario CURRENT OF, che fa parte della clausola where fa sì che non venga trovata alcuna riga da aggiornare. Il cursore dev'essere stato dichiarato in una istruzione DECLARE CURSOR mediante select...for update of.

Se una variabile di set o where è un array, entrambe devono essere array (anche se non si è vincolati sulle dimensioni). Se si tratta di array, update viene eseguito il numero di volte necessario per la gestione di tutti i componenti dell'array. Il numero massimo dipende dalla dimensione dell'array più piccolo o dal valore della clausola for. Vedere FOR per ulteriori dettagli.

UPDATE (forma 2, PL/SQL)

TIPO Istruzione SQL, versione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE DECLARE CURSOR, SOTTOQUERY

SINTASSI UPDATE [utente.]tabella[@dblink]
 SET { colonna = espressione | colonna = (select espressione...) |
 [,colonna = espressione]... |
 [,colonna = (select espressione...)]...] |
 (colonna [,colonna]...) = (sottoquery)}
 [WHERE {condizione | CURRENT OF cursor}];

DESCRIZIONE Il comando UPDATE di PL/SQL funziona in maniera identica a quello di SQL, salvo che per la clausola where opzionale WHERE CURRENT OF cursor. In questo caso l'istruzione update modifica solo la riga su cui si è spostato il cursore a seguito dell'ultimo FETCH. L'istruzione di selezione del cursore deve includere le parole FOR UPDATE.

Come insert, delete e select...into, anche l'istruzione update utilizza sempre il cursore implicito SQL che non viene mai dichiarato (anche se l'istruzione select...for

update deve avere un cursore dichiarato per poter usare la clausola WHERE CURRENT OF cursor). I suoi attributi vengono impostati come segue.

- 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.

ESEMPIO <>overage>>

```
DECLARE
    cursor IMPIEGATO is
        select Eta from LAVORATORE
        for update of Alloggio;
    LAVORATORE_RECORD    IMPIEGATO%rowtype;
BEGIN
    open IMPIEGATO;
    loop
        fetch IMPIEGATO into LAVORATORE_RECORD;
        exit when IMPIEGATO %notfound;
        if LAVORATORE_RECORD.Eta >= 65
            then update LAVORATORE
                set Alloggio = 'YOUTH HOSTEL'
                where current of IMPIEGATO;
        end loop;
        commit;
        close IMPIEGATO;
END overage;
```

UPDATE (forma 3, comando SQL)

TIPO Comando SQL

PRODOTTI Tutti

VEDERE ANCHE DELETE, INSERT, SELECT, SOTTOQUERY, WHERE, Capitolo 13

SINTASSI UPDATE [utente.]tabella[@dblink] [alias]
 SET { colonna = espressione [,colonna = espressione]... |
 (colonna [,colonna]...) = (sottoquery) }
 [WHERE condizione];

DESCRIZIONE UPDATE aggiorna (modifica) i valori delle colonne elencate della tabella specificata. L'opzione where può contenere una sottoquery correlata che può selezionare le righe della tabella da aggiornare (ma una alla volta). Senza alcuna opzione where vengono aggiornate tutte le righe. Le espressioni vengono valutate all'esecuzione del comando e i loro risultati sostituiscono i valori correnti delle celle selezionate.

L'eventuale sottoquery deve selezionare lo stesso numero di colonne (con gli stessi tipi di dati) di quelle presenti tra parentesi sul lato sinistro dell'opzione set.

ESEMPIO Per impostare a NULL le età dei lavoratori con più di 65 anni si utilizza il comando seguente:

```
update LAVORATORE set Eta = NULL where Eta > 65;
```

Il codice seguente aggiorna la Città di Walpole nella tabella COMFORT impostando tutti i valori Precipitazione a NULL, 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

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE LOWER, Capitolo 6

SINTASSI UPPER(*stringa*)

DESCRIZIONE UPPER converte tutti i caratteri di una scritta in lettere maiuscole.

ESEMPIO upper('Guarda che hai fatto, Lauren!')

produce:

GUARDA CHE HAI FATTO, LAUREN!

USER

TIPO Pseudocolonna SQL

PRODOTTI Tutti

VEDERE ANCHE PSEUDOCOLONNE, UID

SINTASSI USER

DESCRIZIONE User è il nome attraverso cui l'utente corrente è riconosciuto da ORACLE. User è una pseudocolonna e come tale può essere consultata da qualsiasi istruzione select ma, non essendo una colonna reale, non può essere aggiornata.

USER PROGRAM INTERFACE (UPI)

L'UPI è quella metà del programma di interfaccia che è responsabile di mantenere separati i programmi utente e il programma di ORACLE per questioni di sicurezza.

USERENV

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI

SINTASSI USERENV(*opzione*)

DESCRIZIONE UserEnv restituisce informazioni relative all'ambiente operativo dell'utente. Le opzioni possibili sono 'ENTRYID', 'SESSIONID' e 'TERMINAL'.

USERNAME

Lo *username* è una parola chiave che identifica un utente di un sistema operativo o di ORACLE. A ciascuno username è sempre associata una password.

VAR (interno SQL)

TIPO Funzione PL/SQL

PRODOTTI PL/SQL

VEDERE ANCHE Vedere SELECT (Forma 2), DICHIARAZIONE DI VARIABILI.

SINTASSI EXEC SQL VAR *variabile_host* IS *tipo_dati*

DESCRIZIONE PL/SQL permette di sovrascrivere l'assegnazione di default di una variabile tramite il comando VAR. All'atto della dichiarazione di una variabile viene assegnato un tipo di dati che può essere sovrascritto da VAR all'interno di un blocco PL/SQL.

VARCHAR

Vedere TIPI DI DATI.

VARCHAR2

Vedere TIPI DI DATI.

VARIABLE

TIPO Funzione PL/SQL

PRODOTTI PL/SQL, SQL*PLUS

VEDERE ANCHE PRINT

SINTASSI VAR[IABLE] [*nome_variabile* {NUMBER|CHAR|CHAR (*n*)}]

DESCRIZIONE VARIABLE dichiara una variabile muta che può essere utilizzata da PL/SQL. A ciascuna variabile viene assegnato un *nome* e un tipo (NUMBER o CHAR). Per le variabili CHAR è possibile specificare una lunghezza massima (*n*).

ESEMPI Nell'esempio seguente viene creata una variabile di nome *bal* il cui valore viene impostato dal risultato di una funzione.

```
variable bal NUMBER
begin
    :bal := CONTROLLO_SALDO('ADAH TALBOT');
end;
```

VARIABILI, DICHIARAZIONE (PL/SQL)

TIPO Istruzione PL/SQL

PRODOTTI PL/SQL

SINTASSI *variabile* [CONSTANT]

```
{tipo | identificativo%TYPE | [utente.]tabella%ROWTYPE}
[NOT NULL]
[DEFAULT | :=] espressione];
```

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 TIPI DI DATI), il tipo di un'altra variabile PL/SQL o quello di una colonna indicata per mezzo di un identificativo o un ROWTYPE (vedere %ROWTYPE) che permette di riferirsi a un record ben definita di una tabella.

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 assegnazione :=) è una qualsiasi espressione PL/SQL da cui risulti un valore del tipo dichiarato.

VARIABILI UTENTE

Vedere ACCEPT, DEFINE, PARAMETRI

VARIANCE

TIPO Funzione di gruppo SQL

PRODOTTI Tutti

VEDERE ANCHE ACCEPT, COMPUTE, DEFINE, FUNZIONI DI GRUPPO, PARAMETERS, STDDEV, Capitolo 7

SINTASSI VARIANCE([DISTINCT] *valore*)

DESCRIZIONE VARIANCE restituisce la varianza statistica di un gruppo di valori provenienti da un gruppo di righe. Come tutte le 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 celle dell'array variabile. *Vedere* il Capitolo 26 per esempi dettagliati riguardanti gli array variabili. *Vedere* CREATE TYPE per informazioni di carattere sintattico.

VERIFY (SQL*PLUS)

Vedere SET.

VERSIONE DI REVISIONE

La versione di revisione è il secondo numero nel codice di versione di ORACLE. Ad esempio, in ORACLE versione 8.0.3, la versione di revisione è 0.

VINCOLO DI INTEGRITÀ

Il vincolo di integrità è una regola che restringe l'intervallo di valori consentiti di una colonna. Viene applicato su una colonna all'atto della sua creazione.

column_constraint *column_constraint* definisce le restrizioni di una colonna e la sua sintassi è la seguente:

```
[CONSTRAINT vincolo]
  {[NOT] NULL |
   [{UNIQUE | PRIMARY KEY}
    [USING INDEX
      [ PCTFREE intero]
      [ INITTRANS intero]
      [ MAXTRANS intero]
      [TABLESPACE tablespace]
      [ STORAGE storage]}}
   [REFERENCES [utente.]tabella[(colonna)] [ON DELETE CASCADE]
   [CHECK (condizione)]]]
```

[EXCEPTIONS INTO *utente.tabella*]
 [DISABLE]

vincolo è un nome che è opzionalmente possibile assegnare al vincolo. Se lo si omette, ORACLE assegna un nome arbitrario del tipo SYS_C_n, dove *n* è un intero. I nomi assegnati automaticamente in genere non rimangono uguali durante importazioni ed esportazioni, 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 unica non può essere dichiarata come PRIMARY KEY (dato che anche PRIMARY KEY forza l'unicità).

REFERENCES identifica la colonna corrente come chiave esterna della colonna *utente.tabella [(colonna)]*. Se si omette la *colonna* ORACLE suppone che il nome di *utente.tabella* sia lo stesso di quello della tabella corrente. Si noti che, quando si utilizza REFERENCES all'interno di un *table_constraint* (esaminato tra breve) questo dev'essere preceduto da FOREIGN KEY. In questo caso non è stato necessario utilizzarlo dato che si tratta dell'unica colonna da indirizzare; *table_constraint* può indicare diverse colonne FOREIGN KEY. ON DELETE CASCADE fa sì che ORACLE mantenga automaticamente l'integrità dei dati rimuovendo le righe contenenti chiavi esterne nelle tabelle dipendenti all'atto della rimozione della 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 booleana (che valga cioè TRUE o FALSE). Può contenere funzioni, colonne appartenenti a tabelle o letterali.

L'opzione EXCEPTIONS INTO specifica la tabella in cui ORACLE mantiene le informazioni relative alle righe che violano le restrizioni di integrità. Questa tabella deve essere locale.

L'opzione DISABLE permette di disabilitare la restrizione di integrità già all'atto della creazione. In seguito sarà possibile abilitare la restrizione attraverso l'opzione ENABLE di ALTER TABLE.

table_constraint *table_constraint* è identico a *column_constraint* tranne che per il fatto che si riferisce a diverse colonne allo stesso tempo; permette ad esempio la dichiarazione di tre colonne come chiavi primarie o esterne. Ecco la sintassi di *table_constraint*:

```
[CONSTRAINT vincolo]
{[NOT] NULL |
[ {UNIQUE | PRIMARY KEY} (colonna[,colonna])
[USING INDEX
  [ PCTFREE intero]
  [ INITTRANS intero]
  [ MAXTRANS intero]
[TABLESPACE tablespace]
```

```
[    STORAGE storage]]]
[FOREIGN KEY (colonna[,colonna])
  REFERENCES [utente.]tabella [ (colonna[,colonna])]
  [ON DELETE CASCADE]]
[CHECK (condizione)]
[EXCEPTIONS INTO [utente.]tabella]
[DISABLE]
```

VINCOLO DI TABELLA

Un vincolo di tabella è un controllo di integrità relativo a diverse colonne della stessa tabella. *Vedere* VINCOLO DI INTEGRITÀ.

VISTA

Una vista è una rappresentazione logica di una tabella. Deriva da una tabella ma non dispone di un proprio spazio di archiviazione. Spesso può essere utilizzata allo stesso modo delle tabelle. *Vedere* CREATE VIEW e il Capitolo 17.

VISTA OGGETTO

Una vista oggetto sovrappone un tipo di dati astratto a una tabella relazionale preesistente. Le viste oggetto permettono di accedere ai dati delle tabelle relazionali sia attraverso normali comandi SQL sia con le strutture dati astratte e forniscono un ponte tecnologico tra database relazionali e database relazionali a oggetti. *Vedere* i Capitoli 25 e 31 per esempi relativi alle viste oggetto.

VISTE DEL DIZIONARIO DI DATI

TIPO Definizioni

PRODOTTI Tutti

VEDERE ANCHE Capitolo 32

DESCRIZIONE Le viste del dizionario dei dati di ORACLE sono descritte nel Capitolo 32. Gran parte dell'elenco seguente, che descrive tutte le viste disponibili del dizionario dei dati, è stato generato selezionando i record dalla vista DICTIONARY in ORACLE, tramite un account non-DBA. Le viste sono elencate qui in ordine alfabetico; nel Capitolo 32 vengono elencate invece in ordine di funzione e ne viene fornita una descrizione.

| NOME DELLA VISTA | COMMENTI |
|------------------------------|---|
| ALL_ARGUMENTS | Argomenti in oggetti accessibili all'utente. |
| ALL_CATALOG | Tutte le tabelle, viste, sinonimi, sequenze accessibili all'utente. |
| ALL_CLUSTERS | Descrizione dei cluster accessibili all'utente. |
| ALL_CLUSTER_HASH_EXPRESSIONS | Funzioni di hash per tutti i cluster accessibili. |
| ALL_COLL_TYPES | Descrizione di tipi di insieme specificati accessibili all'utente. |
| ALL_COL_COMMENTS | Commenti su colonne di tabelle e viste accessibili. |
| ALL_COL_PRIVS | Concessioni sulle colonne per cui l'utente è il beneficiario, il concedente, il proprietario o per cui il beneficiario è un ruolo abilitato o PUBLIC. |

| NOME DELLA VISTA | COMMENTI |
|--------------------------|---|
| ALL_COL_PRIVS_MADE | Concessioni sulle colonne per cui l'utente è il proprietario o il concedente. |
| ALL_COL_PRIVS_REC'D | Concessioni sulle colonne per cui 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 di vincoli. |
| ALL_DB_LINKS | Collegamenti a database accessibili all'utente. |
| ALL_DEF_AUDIT_OPTS | Opzioni di auditing per i nuovi oggetti creati. |
| ALL_DEPENDENCIES | Dipendenze da e verso gli oggetti accessibili all'utente. |
| 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_HISTOGRAMS | Sinonimo per ALL_TAB_HISTOGRAMS. |
| ALL_INDEXES | Descrizioni degli indici sulle tabelle accessibili all'utente. |
| ALL_IND_COLUMNS | Colonne degli indici sulle tabelle accessibili. |
| ALL_IND_PARTITIONS | Partizioni di indici accessibili. |
| ALL_JOBS | Sinonimo per USER_JOB. |
| ALL_LIBRARIES | Descrizione delle librerie accessibili all'utente. |
| ALL_LOBS | Descrizione dei LOB contenuti nelle tabelle accessibili all'utente. |
| 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_NESTED_TABLES | Descrizione delle tabelle annidate nelle tabelle accessibili all'utente. |
| ALL_OBJECTS | Oggetti accessibili all'utente. |
| ALL_PART_COL_STATISTICS | Informazioni statistiche per le colonne di partizioni. |
| ALL_PART_HISTOGRAMS | Informazioni statistiche per le partizioni. |
| ALL_PART_INDEXES | Indici per le partizioni accessibili. |
| ALL_PART_KEY_COLUMNS | Colonne chiave utilizzate per la suddivisione in partizioni. |
| ALL_PART_TABLES | Tutte le tabelle di partizioni accessibili. |
| ALL_REFRESH | Tutti i gruppi di refresh a cui l'utente può accedere. |
| ALL_REFRESH_CHILDREN | Tutti gli oggetti nei gruppi di refresh, dove l'utente può accedere al gruppo. |
| ALL_REFS | Descrizione di colonne REF contenute in tabelle accessibili all'utente. |
| ALL_REGISTERED_SNAPSHOTS | Snapshot remoti di tabelle locali che l'utente può vedere. |
| ALL_SEQUENCES | Descrizione delle sequenze accessibili all'utente. |
| ALL_SNAPSHOTS | Snapshot che l'utente può visualizzare. |

| NOME DELLA VISTA | COMMENTI |
|----------------------------|---|
| ALL_SNAPSHOT_LOGS | Tutti i log di snapshot nel database, che l'utente può vedere. |
| ALL_SNAPSHOT_REFRESH_TIMES | Snapshot e relativi tempi di aggiornamento o ciascuna tabella master che l'utente può guardare. |
| ALL_SOURCE | Sorgente corrente sugli oggetti memorizzati che l'utente può creare. |
| ALL_SYNONYMS | Tutti i sinonimi accessibili all'utente. |
| ALL_TABLES | Descrizione delle tabelle accessibili all'utente. |
| 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 | Iistogrammi sulle colonne di tutte le tabelle visibili all'utente. |
| ALL_TAB_PARTITIONS | Tutte le tabelle che sono state suddivise in partizioni. |
| ALL_TAB_PRIVS | Concessioni sugli oggetti per cui l'utente è il beneficiario, il concedente, il proprietario o per cui 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 l'utente a cui è stato garantito il ruolo PUBLIC o è stato abilitato il ruolo . |
| ALL_TRIGGERS | Trigger accessibili all'utente corrente. |
| ALL_TRIGGER_COLS | Utilizzo delle colonne nei trigger dell'utente o nei trigger associati alle colonne 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_UPDATABLE_COLUMNS | Descrizione di tutte le colonne aggiornabili. |
| ALL_USERS | Informazioni relative a tutti gli utenti del database. |
| ALL_VIEWS | Descrizione delle viste accessibili all'utente. |
| AUDIT_ACTIONS | Tabella descrittiva per i codici dei tipi di azione per la coda di auditing. Associa i numeri dei tipi di azione con i nomi delle azioni. |
| CAT | Sinonimo per USER_CATALOG. |
| CLU | Sinonimo per USER_CLUSTERS. |
| COLS | Sinonimo per USER_TAB_COLUMNS. |
| COLUMN_PRIVILEGES | Concessioni sulle colonne per cui l'utente è il beneficiario, il concedente, il proprietario o per cui il beneficiario è un ruolo abilitato o PUBLIC. |
| DICT | Sinonimo per DICTIONARY. |
| DICTIONARY | Descrizione delle tabelle e delle viste del dizionario dei dati. |
| DICT_COLUMNS | Descrizione delle colonne nelle tabelle e nelle viste del dizionario dei dati. |

| NOME DELLA VISTA | COMMENTI |
|-------------------------------|---|
| DUAL | Tabella di una riga e una colonna, utilizzata per i test logici. |
| GLOBAL_NAME | Nome globale del database. |
| IND | Sinonimo per USER_INDEXES. |
| INDEX_HISTOGRAM | Informazioni statistiche sulle chiavi con conteggio 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 che sono concessi a altri ruoli. |
| ROLE_SYS_PRIVS | Privilegi di sistema concessi ai ruoli. |
| ROLE_TAB_PRIVS | Privilegi sulle tabelle 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 cui l'utente è il beneficiario, il concedente, il proprietario o per cui il beneficiario è un ruolo abilitato o PUBLIC. |
| TABS | Sinonimo per USER_TABLES. |
| USER_ARGUMENTS | Argomenti in un oggetto accessibili all'utente. |
| USER_AUDIT_OBJECT | Record di code di auditing che riguardano oggetti, in modo particolare: tabelle, cluster, viste, indici, sequenze, collegamenti a database [public], sinonimi [public], procedure, trigger, segmenti di rollback, tablespace, ruoli, utenti |
| USER_AUDIT_SESSION | Tutti i record delle code di auditing che riguardano CONNECT e DISCONNECT. |
| USER_AUDIT_STATEMENT | Record della coda di auditing che riguardano i comandi grant, revoke, audit, noaudit e alter system. |
| USER_AUDIT_TRAIL | Voci della coda di auditing rilevanti per l'utente. |
| 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 di hash dell'utente. |
| USERCLU_COLUMNS | Corrispondenza tra le colonne della tabella e quelle del cluster. |
| USER_COLL_TYPES | Descrizione dei tipi di insiemi specificati di proprietà dell'utente. |

| NOME DELLA VISTA | COMMENTI |
|--------------------------|--|
| USER_COL_COMMENTS | Commenti sulle colonne delle tabelle e delle viste dell'utente. |
| USER_COL_PRIVS | Concessioni sulle colonne per cui l'utente è il proprietario, il concedente o il beneficiario. |
| USER_COL_PRIVS_MADE | Tutti i permessi sulle colonne degli oggetti di proprietà dell'utente. |
| USER_COL_PRIVS_REC'D | Permessi sulle colonne per cui 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 | Collegamenti a database di proprietà dell'utente. |
| USER_DEPENDENCIES | Dipendenze verso e dagli oggetti di un utente. |
| USER_ERRORS | Errori correnti sugli oggetti memorizzati di proprietà dell'utente. |
| USER_EXTENTS | Estensioni comprendenti segmenti di proprietà dell'utente. |
| USER_FREE_SPACE | Estensioni libere nei tablespace accessibili all'utente. |
| USER_HISTOGRAMS | Sinonimo per USER_TAB_HISTOGRAMS. |
| USER_INDEXES | Descrizione degli indici di proprietà dell'utente. |
| USER_IND_COLUMNS | Colonne comprendenti indici dell'utente o su tabelle dell'utente. |
| USER_IND_PARTITIONS | Partizioni di indici di proprietà di un utente. |
| USER_JOBS | Tutti i job di proprietà di un utente. |
| USER_LIBRARIES | Descrizione delle librerie di proprietà dell'utente. |
| USER_LOBS | Descrizione dei LOB di proprietà dell'utente contenuti nelle tabelle dell'utente. |
| USER_METHOD_PARAMS | Descrizione dei parametri del metodo dei tipi di proprietà dell'utente. |
| USER_METHOD_RESULTS | Descrizione dei risultati del metodo dei tipi di proprietà dell'utente. |
| 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, dei vari oggetti di PL/SQL. |
| USER_OBJ_AUDIT_OPTS | Opzioni di auditing per le tabelle e le viste di proprietà dell'utente. |
| USER_PART_COL_STATISTICS | Informazioni statistiche sulle colonne per le partizioni dell'utente. |
| USER_PART_HISTOGRAMS | Iistogrammi dei dati per le partizioni dell'utente. |
| USER_PART_INDEXES | Indici per le partizioni dell'utente. |
| USER_PART_KEY_COLUMNS | Colonne chiave utilizzate per suddividere in partizioni i dati dell'utente. |
| USER_PART_TABLES | Tabelle della partizione dell'utente. |
| USER_PASSWORD_LIMITS | Visualizza i limiti della password dell'utente. |
| USER_REFRESH | Tutti i gruppi di aggiornamento. |
| USER_REFRESH_CHILDREN | Tutti gli oggetti nei gruppi di aggiornamento per cui l'utente è il proprietario del gruppo. |

| NOME DELLA VISTA | COMMENTI |
|-----------------------------|--|
| USER_REFS | Descrizione delle colonne REF di proprietà dell'utente contenute nelle tabelle dell'utente. |
| USER_REGISTERED_SNAPSHOTS | Snapshot remoti delle tabelle locali che attualmente utilizzano log di proprietà dell'utente. |
| USER_RESOURCE_LIMITS | Visualizza i limiti delle risorse dell'utente. |
| USER_ROLE_PRIVS | Ruoli concessi all'utente corrente. |
| USER_SEGMENTS | Memoria allocata per tutti i segmenti del database. |
| USER_SEQUENCES | Descrizione delle sequenze di proprietà dell'utente. |
| USER_SNAPSHOTS | Snapshot che l'utente può visualizzare. |
| USER_SNAPSHOT_LOGS | Tutti i log degli snapshot di proprietà dell'utente. |
| USER_SNAPSHOT_REFRESH_TIMES | Snapshot e relativi ultimi tempi di aggiornamento per ciascuna tabella master che l'utente può visualizzare. |
| USER_SOURCE | Sorgente degli oggetti memorizzati accessibili all'utente. |
| USER_SYNONYMS | Sinonimi privati dell'utente. |
| USER_SYS_PRIVS | Privilegi di sistema concessi all'utente corrente. |
| USER_TABLES | Descrizione delle tabelle di proprietà dell'utente. |
| USER_TABLESPACES | Descrizione dei tablespace accessibili. |
| 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_PARTITIONS | Tabelle dell'utente che sono state suddivise in partizioni. |
| USER_TAB_PRIVS | Privilegi sugli oggetti per i quali l'utente è proprietario, concedente o beneficiario. |
| USER_TAB_PRIVS_MADE | Tutti i privilegi sugli oggetti di proprietà dell'utente. |
| USER_TAB_PRIVS_REC'D | Permessi sugli oggetti per i quali l'utente è beneficiario. |
| USER_TRIGGERS | Trigger di proprietà dell'utente. |
| USER_TRIGGER_COLS | Utilizzo 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_UPDATABLE_COLUMNS | Descrizione delle colonne aggiornabili. |
| USER_USERS | Informazioni relative all'utente corrente. |
| USER_VIEWS | Descrizione delle viste di proprietà dell'utente. |
| V\$ACTIVE_INSTANCES | Sinonimo per V_\$ACTIVE_INSTANCES. |

| NOME DELLA VISTA | COMMENTI |
|---------------------|---|
| V\$LOADCSTAT | Sinonimo per V_\$LOADCSTAT. |
| V\$LOADPSTAT | Sinonimo per V_\$LOADPSTAT. |
| V\$LOADTSTAT | Sinonimo per V_\$LOADTSTAT. |
| V\$MLS_PARAMETERS | Sinonimo per V_\$MLS_PARAMETERS. |
| V\$NLS_PARAMETERS | Sinonimo per V_\$NLS_PARAMETERS. |
| V\$NLS_VALID_VALUES | Sinonimo per V_\$NLS_VALID_VALUES. |
| V\$OPTION | Sinonimo per V_\$OPTION. |
| V\$PQ_SESSTAT | Informazioni statistiche relative alla Parallel Query Option. |
| V\$PQ_TQSTAT | Informazioni statistiche relative alla Parallel Query Option. |
| V\$SESSION_LONGOPS | Sinonimo per V_\$SESSION_LONGOPS. |
| V\$VERSION | Informazioni sulla versione del database. |

VIRTUALE, COLONNA

Una colonna virtuale è il 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.

VSIZE

TIPO Funzione SQL

PRODOTTI Tutti

VEDERE ANCHE CARATTERI, FUNZIONI; LENGTH; NUMERI, FUNZIONI; Capitolo 7

SINTASSI VSIZE(*valore*)

DESCRIZIONE VSIZE restituisce il numero di byte necessari per l'archiviazione del *valore* in ORACLE. Per le colonne di caratteri VSIZE è esattamente equivalente a LENGTH. Per i numeri è generalmente minore della lunghezza apparente dato che i database dispongono di routine ottimizzate per l'archiviazione dei numeri.

ESEMPI VSIZE('VICTORIA') = 8

VSIZE(12.345) = 4

WHENEVER (forma 1, interno SQL)

TIPO Comando SQL interno

PRODUCT Precompilatori

VEDERE ANCHE EXECUTE, FETCH

SINTASSI EXEC SQL WHENEVER {NOT FOUND | SQLERROR | SQL WARNING}
{CONTINUE |
GOTO *etichetta* |
STOP |
DO *routine*}

DESCRIZIONE WHENEVER non è un'istruzione eseguibile di SQL, ma piuttosto una direttiva di ORACLE che di fatto aggiunge una condizione del tipo “IF *condizione* THEN GOTO *etichetta*” al termine di ogni istruzione seguente.

WHENEVER può essere utilizzato con CONTINUE o con un'etichetta diversa sulla base di un massimo di tre 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.

La condizione NOT FOUND viene considerata veraognualvolta che SQLCODE valga 100, come ad esempio quando un FETCH non restituisce nessuna riga (compatibilmente anche con le sottoquery delle istruzioni insert).

SQLERROR è vera quando SQLCODE è minore di 0. Si tratta generalmente di errori fatali che richiedono pesanti routine di gestione.

SQLWARNING è vera quando si verifica un errore non fatale. In questa categoria ricadono il troncamento di stringhe di caratteri caricate all'interno di variabili host, l'esecuzione di istruzioni di cancellazione o aggiornamento prive di clausole where e così via.

In genere queste istruzioni vengono impostate all'inizio del programma, prima di qualsiasi istruzione eseguibile SQL in maniera analoga alla seguente:

```
exec sql whenever not found goto close_down;
exec sql whenever sqlwarning continue;
exec sql whenever sqlerror goto fatal_error;
```

Comunque, all'interno di particolari blocchi logici del programma, esse possono essere sovrascritte in base alle necessità locali. Si noti che WHENEVER non è al corrente di linguaggi host, chiamate, procedure ecc. Dal momento dell'esecuzione di WHENEVER il suo effetto rimane il medesimo fino al successivo WHENEVER (che usi la stessa condizione).

L'opzione STOP interrompe l'esecuzione del programma; l'opzione DO richiama una routine di qualche tipo tra quelle del linguaggio host.

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 richiamare una routine di uscita in modo da evitare cicli infiniti.

WHENEVER SQLERROR (forma 2, SQL*PLUS)

TIPO Comando SQL, versione modificata di WHENEVER

PRODUCT SQL*PLUS

VEDERE ANCHE EXIT, WHENEVER (forma 1)

SINTASSI WHENEVER SQLERROR {EXIT
[SUCCESS|FAILURE|WARNING|*intero|variabile*] |
CONTINUE}

DESCRIZIONE Con EXIT, WHENEVER SQLERROR conclude una sessione SQL*PLUS e qualora riscontri un errore fatale (con sqlcode minore di zero) riporta al programma chiamante o al sistema operativo. Non sono compresi i codici di pericolo come “nessuna riga selezionata” o “aggiornamento senza alcuna clausola” mentre lo sono quelli del tipo “la tabella o vista non esiste”. Scrive ogni eventuale modifica pendente sul database. Viene inoltre restituito un codice di ritorno. SUCCESS, FAILURE e WARNING hanno valori che variano da un sistema operativo all'altro. FAILURE e WARNING possono coincidere su alcune piattaforme.

Con **CONTINUE** (invece di **EXIT**), gli errori SQL vengono ignorati e viene continuata l'esecuzione delle istruzioni successive. Questo può essere impiegato all'interno dei file di avvio, quando alcuni resoconti il cui risultato sia indipendente dal successo o meno delle istruzioni SQL precedenti debbano comunque essere completati.

È possibile porre **WHENEVER SQLERROR** in un file di avvio davanti a un numero qualsiasi di istruzioni. Ogni **WHENEVER SQLERROR** sovrascrive il precedente e rimane attivo fino a che non ne interviene uno successivo a sovrascriverlo.

intero è il valore che si vuole passare al programma chiamante come codice di ritorno; *variabile* permette di impostare questo valore dinamicamente. Può essere una variabile utente o una variabile di sistema come **sql.sqlcode** che contiene sempre il codice sql dell'ultima istruzione SQL eseguita, sia in **SQL*PLUS** che in un blocco **PL/SQL** interno.

ESEMPIO Nell'esempio seguente **create table** genera un errore (dato che il letterale **KEENE** non è racchiuso tra apici), e i comandi **update** e **select** non vengono mai eseguiti. **WHENEVER SQLERROR** esce immediatamente da **SQL*PLUS** e passa il codice **sql.sqlcode** al programma chiamante o al sistema operativo:

```
whenever sqlerror exit sql.sqlcode;

create table KEENE as
select * from COMFORT
where Citta = KEENE;

update KEENE set Mezzogiorno = 35;

select * from KEENE;
```

WHERE

TIPO Clausola SQL

PRODOTTI Tutti

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 ritornare, cancellare o aggiornare.

Una condizione può essere definita in uno dei modi seguenti.

- Un confronto tra espressioni (=, >, != 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 una 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 Importo >= Tasso * Quantita;
      where Articolo = 'MEDISON';
      where ('GEORGE','VICTORIA') = (select Marito, Moglie from
COPPIA);
      where Sezione >ANY ('A','C','D');
      where Citta !=ALL (select Citta from LOCAZIONE);
      where Sezione IN ('A','C','D');
      where Citta NOT IN (select Citta from LOCAZIONE);
      where Pag BETWEEN 4 and 7;
      where Compito IS NULL;
      where EXISTS (select * from LAVORATORE where Eta > 90);
      where Sezione = 'A' or Sezione = 'B' and Pag = 1;
```

WHILE in PL/SQL

Vedere CICLI, Capitolo 22.

WRAP (SQL*PLUS)

Vedere SET.

WRAPPING

Wrapping significa spezzare un titolo o una colonna portandolo su un'altra riga quando viene superato il margine di visualizzazione. È l'effetto opposto di TRUNCATE. Vedere COLUMN.

- Parte sesta
 - Appendici**

•
•
•
•
• Appendice A

• **Le tavole utilizzate nel libro**

•
•
•
•
• A.1 **Utilizzo delle tavole**

In questa appendice sono riportati i listati completi di tutte le tavole utilizzate negli esempi riportati nel presente volume. Possono essere digitati in un file di avvio per caricarli in un database con SQLPLUS. Una volta inserite queste tavole nel proprio database ORACLE, risulta più semplice seguire gli esempi riportati nel libro e sperimentare le molte tecniche illustrate.

Per semplicità di utilizzo, conviene impostare un utente di nome “esercizi”, con password “esercizi” e autorità CONNECT e RESOURCE, in modo che nuovi utenti possano accedervi per apprendere l’uso di ORACLE. Se è la prima volta che si lavora con ORACLE conviene chiedere al DBA se esiste già un nome utente di questo tipo, in caso contrario lo si può richiedere, o almeno richiedere che tavole e file di avvio siano resi disponibili tramite sinonimi pubblici. Qualunque “danno” sia eventualmente apportato all’identificativo “esercizi” può sempre essere riparato ricaricando le tavole dai file di avvio.

A.1 Utilizzo delle tavole

Le tavole di questa appendice si trovano anche sul CD allegato, perciò non è necessario digitare i listati. Si possono utilizzare queste tavole per seguire meglio gli esempi e sperimentare le molte tecniche illustrate nel libro.

Il file di avvio activity.sql

```
rem      Nome: activity.sql    Tipo: file di avvio report
rem  Scritto da: G. Koch
rem
rem Descrizione: Report delle vendite per prodotto di G. B. Talbot
rem                  durante la seconda metà del 1901.
```

```
set headsep !
```

```
ttitle 'Vendite per prodotto durante il 1901!Secondo semestre (lug-dic)'
btitle 'dal registro di G. B. Talbot'
```

```
column Articolo heading 'Bene!Venduto'
column Articolo format a18
column Articolo truncated
column Persona heading 'Venduto a' format a18 word_wrapped
column Tasso format 90.99
column DataAzione heading 'Data'
column TipoQuantita heading 'Tipo' format a8 truncated
column TipoQuantita heading 'Quan' format 9990
column Pro format 990.99
break on Articolo skip 2
compute sum of Pro on Articolo

set linesize 79
set pagesize 50
set newpage 0

spool activity.1st

select DataAzione, Persona, Articolo, Quantita, TipoQuantita,
       Tasso, Quantita * Tasso "Pro"
  from REGISTRO
 where Azione = 'VENDUTO'          /* solo negli ultimi 6 mesi */
       and DataAzione BETWEEN TO_DATE('01-JUL-1901','DD-MON-YYYY')
       and TO_DATE('31-DEC-1901','DD-MON-YYYY')
order by Articolo, DataAzione;

spool off
```

Il file di avvio math.sql

```
rem      Nome: math.sql      Tipo: file di avvio report
rem  Scritto da: G. Koch
rem
rem  Descrizione: Illustra le funzioni matematiche di ORACLE. Produce
rem                  un file di output denominato MATH.LST
set echo off
set pagesize 32000
set linesize 132
column Piu format 99.999
column Meno format 999.999
column Per format 9999.999999
column Diviso format .999999
column VSIZE(Sopra) heading 'VSIZE|(Sopra)'
column VSIZE(Sotto) heading 'VSIZE|(Sotto)'
column VSIZE(Vuoto) heading 'VSIZE|(Vuoto)'
column NVL(Sopra,11) heading 'NVL|(Sopra,11)'
column NVL(Sotto,11) heading 'NVL|(Sotto,11)'
column NVL(Vuoto,11) heading 'NVL|(Vuoto,11)'
column ROUND(Sopra,2) heading 'ROUND|(Sopra,2)'
column ROUND(Sotto,2) heading 'ROUND|(Sotto,2)'
column TRUNC(Sopra,2) heading 'TRUNC|(Sopra,2)'
```

```
column TRUNC(Sotto,2) heading 'TRUNC|(Sotto,2)'
column ROUND(Sopra,0) heading 'ROUND|(Sopra,0)'
column ROUND(Sotto,0) heading 'ROUND|(Sotto,0)'
column TRUNC(Sopra,0) heading 'TRUNC|(Sopra,0)'
column TRUNC(Sotto,0) heading 'TRUNC|(Sotto,0)'
column ROUND(Sopra,-1) heading 'ROUND|(Sopra,-1)'
column ROUND(Sotto,-1) heading 'ROUND|(Sotto,-1)'
column TRUNC(Sopra,-1) heading 'TRUNC|(Sopra,-1)'
column TRUNC(Sotto,-1) heading 'TRUNC|(Sotto,-1)'
set echo on
set numwidth 7
set numformat ""
spool math.lst
select Nome, Sopra, Sotto, Vuoto from MATEMATICA;

select Nome, Sopra, Sotto, Vuoto,
       Sopra + Sotto "Piu",
       Sopra - Sotto "Meno",
       Sopra * Sotto "Per",
       Sopra / Sotto "Diviso"
  from MATEMATICA where Nome = 'DECIMALE SUP';

select Nome, Sopra, Sotto, Vuoto,
       Sopra + Vuoto "Piu",
       Sopra - Vuoto "Meno",
       Sopra * Vuoto "Per",
       Sopra / Vuoto "Diviso"
  from MATEMATICA where Nome = 'DECIMALE SUP';

select Nome, Sopra, Sotto, Vuoto,
       ABS(Sopra),
       ABS(Sotto),
       ABS(Vuoto)
  from MATEMATICA where Nome = 'DECIMALE SUP';

select Nome, Sopra, Sotto, Vuoto,
       VSIZE(Sopra),
       VSIZE(Sotto),
       VSIZE(Vuoto)
  from MATEMATICA;

select Nome, Sopra, Sotto, Vuoto,
       NVL(Sopra,11),
       NVL(Sotto,11),
       NVL(Vuoto,11)
  from MATEMATICA where Nome = 'DECIMALE SUP';

select Nome, Sopra, Sotto,
       CEIL(Sopra),
       CEIL(Sotto)
  from MATEMATICA;
```

```
select Nome, Sopra, Sotto,
       FLOOR(Sopra),
       FLOOR(Sotto)
  from MATEMATICA;

select Nome, Sopra, Sotto,
       MOD(Sopra,11),
       MOD(Sotto,11)
  from MATEMATICA where Nome = 'DECIMALE SUP';

set numformat 9999.99999
select Nome, Sopra, Sotto,
       POWER(Sopra,2),
       POWER(Sotto,2)
  from MATEMATICA where Nome = 'DECIMALE SUP';

select Nome, Sopra, Sotto,
       POWER(Sopra,2.9),
       POWER(Sotto,2.9)
  from MATEMATICA;

set numformat ""
select Nome, Sopra, Sotto,
       SQRT(Sopra),
       SQRT(Sotto)
  from MATEMATICA where Nome = 'DECIMALE SUP';

select Nome, Sopra, Sotto,
       ROUND(Sopra,2),
       ROUND(Sotto,2),
       TRUNC(Sopra,2),
       TRUNC(Sotto,2)
  from MATEMATICA;

select Nome, Sopra, Sotto,
       ROUND(Sopra,0),
       ROUND(Sotto,0),
       TRUNC(Sopra,0),
       TRUNC(Sotto,0)
  from MATEMATICA;

select Nome, Sopra, Sotto,
       ROUND(Sopra,-1),
       ROUND(Sotto,-1),
       TRUNC(Sopra,-1),
       TRUNC(Sotto,-1)
  from MATEMATICA;

select Nome, Sopra, Sotto,
       SIGN(Sopra),
       SIGN(Sotto)
  from MATEMATICA where Nome = 'DECIMALE SUP';
spool off
```

Il tipo di dati INDIRIZZO_TY

```
rem Richiede l'uso di ORACLE8 Objects.
```

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

Il tipo di dati ANIMALE_TY

```
rem Richiede l'uso di ORACLE8 Objects.
```

```
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));
create or replace type body ANIMALE_TY as
member function Eta (DataNascita DATE) return NUMBER is
begin
  RETURN ROUND(SysDate - DataNascita);
end;
end;
/
```

```
rem per gli esempi di tabella annidata:
```

```
create type ANIMALI_NT as table of ANIMALE_TY;
```

Il tipo di dati PERSONA_TY

```
rem Richiede l'uso di ORACLE8 Objects.
```

```
rem Richiede che il tipo di dati INDIRIZZO_TY esista già.
```

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

Il tipo di dati ATTREZZI_VA

```
rem Richiede l'uso di ORACLE8.
```

```
rem Crea un tipo di dati array variabile.
```

```
create or replace type ATTREZZI_VA as varray(5) of VARCHAR2(25);
```

La tabella ALLEVAMENTO

```
drop table ALLEVAMENTO;
create table ALLEVAMENTO (
Figlio          VARCHAR2(10),
Sex             CHAR(1),
Mucca          VARCHAR2(10),
Toro            VARCHAR2(10),
Datanascita    DATE
);

insert into ALLEVAMENTO values ('EVE','F',null,null,null);
insert into ALLEVAMENTO values ('ADAM','M',null,null,null);
insert into ALLEVAMENTO values ('BANDIT','M',null,null,null);
insert into ALLEVAMENTO values ('BETSY','F','EVE','ADAM',
      TO_DATE('02-JAN-1900','DD-MON-YYYY'));
insert into ALLEVAMENTO values ('POCO','M','EVE','ADAM',
      TO_DATE('15-JUL-1900','DD-MON-YYYY'));
insert into ALLEVAMENTO values ('GRETA','F','EVE','BANDIT',
      TO_DATE('12-MAR-1901','DD-MON-YYYY'));
insert into ALLEVAMENTO values ('MANDY','F','EVE','POCO',
      TO_DATE('22-AUG-1902','DD-MON-YYYY'));
insert into ALLEVAMENTO values ('NOVI','F','BETSY','ADAM',
      TO_DATE('30-MAR-1903','DD-MON-YYYY'));
insert into ALLEVAMENTO values ('GINNY','F','BETSY','BANDIT',
      TO_DATE('04-DEC-1903','DD-MON-YYYY'));
insert into ALLEVAMENTO values ('CINDY','F','EVE','POCO',
      TO_DATE('09-FEB-1903','DD-MON-YYYY'));
insert into ALLEVAMENTO values ('DUKE','M','MANDY','BANDIT',
      TO_DATE('24-JUL-1904','DD-MON-YYYY'));
insert into ALLEVAMENTO values ('TEDDI','F','BETSY','BANDIT',
      TO_DATE('12-AUG-1905','DD-MON-YYYY'));
insert into ALLEVAMENTO values ('SUZY','F','GINNY','DUKE',
      TO_DATE('03-APR-1906','DD-MON-YYYY'));
insert into ALLEVAMENTO values ('RUTH','F','GINNY','DUKE',
      TO_DATE('25-DEC-1906','DD-MON-YYYY'));
insert into ALLEVAMENTO values ('PAULA','F','MANDY','POCO',
      TO_DATE('21-DEC-1906','DD-MON-YYYY'));
insert into ALLEVAMENTO values ('DELLA','F','SUZY','BANDIT',
      TO_DATE('11-OCT-1908','DD-MON-YYYY'));
```

La tabella ALLOGGIO

```
drop table ALLOGGIO;
create table ALLOGGIO (
Alloggio        VARCHAR2(15) not null,
NomeLungo       VARCHAR2(40),
Direttore       VARCHAR2(25),
Indirizzo       VARCHAR2(30)
);
```

```

insert into ALLOGGIO values (
'CRANMER','CRANMER RETREAT HOUSE','THOM CRANMER','HILL ST, BERKELEY');
insert into ALLOGGIO values (
'MATTS','MATTS LONG BUNK HOUSE','ROLAND BRANDT','3 MILE RD, KEENE');
insert into ALLOGGIO values (
'MULLERS','MULLERS COED ALLOGGIO','KEN MULLER','120 MAIN, EDMESTON');
insert into ALLOGGIO values (
'PAPA KING','PAPA KING ROOMING','WILLIAM KING','127 MAIN, EDMESTON');
insert into ALLOGGIO values (
'ROSE HILL','ROSE HILL FOR MEN','JOHN PELETIER','RFD 3, N. EDMESTON');
insert into ALLOGGIO values (
'WEITBROCHT','WEITBROCHT ROOMING','EUNICE BENSON','320 GENEVA,
KEENE');

```

La tabella AREA

```

drop table AREA;
create table AREA
(Radius      NUMBER(5),
 Area        NUMBER(14,2));

```

REMARK Nessun record. Questa tabella è vuota.

La tabella AZIONE

```

drop table AZIONE;
create table AZIONE (
Societa      VARCHAR2(20),
Simbolo      VARCHAR2(6),
Industria    VARCHAR2(15),
ChiusIeri    NUMBER(6,2),
ChiusOggi    NUMBER(6,2),
Volume       NUMBER
);
insert into AZIONE values (
'AD SPECIALTY'      , 'ADSP', 'PUBBLICITA',     31.75,      31.75,
18333876);
insert into AZIONE values (
'APPLE CANNERY'     , 'APCN', 'AGRICOLTURA',   33.75,      36.50,
25787229);
insert into AZIONE values (
'AT SPACE'          , 'ATR ', 'MEDICALE',      46.75,      48.00,
11398323);
insert into AZIONE values (
'AUGUST ENTERPRISES', 'AGE', 'MEDICALE',      15.00,      15.00,
12221711);
insert into AZIONE values (
'BRANDON ELLIPSIS'   , 'BELP', 'SPAZIO',        32.75,      33.50,
25789769);

```

```
insert into AZIONE values (
'GENERAL ENTROPY'      , 'GENT', 'SPAZIO',           64.25,   66.00,
    7598562);
insert into AZIONE values (
'GENEVA ROCKETRY'     , 'GENR', 'SPAZIO',           22.75,   27.25,
    22533944);
insert into AZIONE values (
'HAYWARD ANTISEPTIC'  , 'HAYW', 'MEDICALE',        104.25,  106.00,
    3358561);
insert into AZIONE values (
'IDK'                  , 'IDK', 'ELETTRONICA',       95.00,   95.25,
    9443523);
insert into AZIONE values (
'INDIA COSMETICS'     , 'INDI', 'COSMETICA',       30.75,   30.75,
    8134878);
insert into AZIONE values (
'ISAIAH JAMES STORAGE', 'IJS', 'TRASPORTI',        13.25,   13.75,
    22112171);
insert into AZIONE values (
'KDK AIRLINES'         , 'KDK', 'TRASPORTI',        85.00,   85.25,
    7481566);
insert into AZIONE values (
'KENTGEN BIOPHYSICS'   , 'KENT', 'MEDICALE',        18.25,   19.50,
    6636863);
insert into AZIONE values (
'LAVAY COSMETICS'      , 'LAVA', 'COSMETICA',       21.50,   22.00,
    3341542);
insert into AZIONE values (
'LOCAL DEVELOPMENT'    , 'LOCD', 'AGRICOLTURA',      26.75,   27.25,
    2596934);
insert into AZIONE values (
'MAXTIDE'               , 'MAXT', 'TRASPORTI',        8.25,    8.00,
    2836893);
insert into AZIONE values (
'MBK COMMUNICATIONS'   , 'MBK', 'PUBBLICITA',       43.25,   41.00,
    10022980);
insert into AZIONE values (
'MEMORY GRAPHICS'      , 'MEMG', 'ELETTRONICA',       15.50,   14.25,
    4557992);
insert into AZIONE values (
'MICRO TOKEN'          , 'MICT', 'ELETTRONICA',       77.00,   76.50,
    25205667);
insert into AZIONE values (
'NANCY LEE FEATURES'   , 'NLF', 'PUBBLICITA',       13.50,   14.25,
    14222692);
insert into AZIONE values (
'NORTHERN BOREAL'      , 'NBOR', 'SPAZIO',           26.75,   28.00,
    1348323);
insert into AZIONE values (
'OCKHAM SYSTEMS'        , 'OCKS', 'SPAZIO',           21.50,   22.00,
    7052990);
```

```

insert into AZIONE values (
'OSCAR COAL DRAYAGE' , 'OCD','TRASPORTI',      87.00,     88.50,
25798992);
insert into AZIONE values (
'ROBERT JAMES APPAREL','RJAP','VESTIARIO',    23.25,     24.00,
19032481);
insert into AZIONE values (
'SOUP SENSATIONS'   , 'SOUP','AGRICOLTURA',   16.25,     16.75,
22574879);
insert into AZIONE values (
'WONDER LABS'       , 'WOND','SPAZIO',        5.00,      5.00,
2553712);

```

La tabella BUONO

```

drop table BUONO;
create table BUONO (
Conto      NUMBER not null,
Importo    NUMBER not null,
DataMaturazione DATE not null
);

insert into BUONO values (573334, 10000,
TO_DATE('15-JAN-2009','DD-MON-YYYY'));
insert into BUONO values (677654, 25000,
TO_DATE('15-JAN-2001','DD-MON-YYYY'));
insert into BUONO values (976032, 10000,
TO_DATE('15-JAN-1995','DD-MON-YYYY'));
insert into BUONO values (275031, 10000,
TO_DATE('15-JAN-1997','DD-MON-YYYY'));
insert into BUONO values (274598, 20000,
TO_DATE('15-JAN-1999','DD-MON-YYYY'));
insert into BUONO values (538365, 45000,
TO_DATE('15-JAN-2001','DD-MON-YYYY'));
insert into BUONO values (267432, 16500,
TO_DATE('15-JAN-2004','DD-MON-YYYY'));

```

La tabella CLIENTE

```

rem Richiede l'utilizzo di ORACLE8 Objects.
rem Richiede l'esistenza dei tipi di dati INDIRIZZO_TY e PERSONA_TY.

create table CLIENTE
(Cliente_ID    NUMBER,
Persona        PERSONA_TY);

insert into CLIENTE
(1,
PERSONA_TY('NEIL MULLANE',

```

```
INDIRIZZO_TY('57 MT PLEASANT ST',
              'FINN', 'NH', 11111));

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

La tabella CLIMA

```
drop table CLIMA;
create table CLIMA (
Citta      VARCHAR2(11),
Temperatura NUMBER,
Umidita    NUMBER,
Condizione VARCHAR2(9)
);

insert into CLIMA values ('LIMA',45,79,'PIOGGIA');
insert into CLIMA values ('PARIGI',81,62,'NUVOLOSO');
insert into CLIMA values ('MANCHESTER',66,98,'NEBBIA');
insert into CLIMA values ('ATENE',97,89,'SOLE');
insert into CLIMA values ('CHICAGO',66,88,'PIOGGIA');
insert into CLIMA values ('SYDNEY',29,12,'NEVE');
insert into CLIMA values ('SPARTA',74,63,'NUVOLOSO');
```

La tabella COMFORT

```
drop table COMFORT;
create table COMFORT (
Citta      VARCHAR2(13) NOT NULL,
DataCampione DATE NOT NULL,
Mezzogiorno NUMBER(3,1),
Mezzanotte NUMBER(3,1),
Precipitazione NUMBER
);

SAN FRANCISCO 21-MAR-93      16.9      5.7
SAN FRANCISCO 22-JUN-93      10.6     22.2
SAN FRANCISCO 23-SEP-93          16.4
SAN FRANCISCO 22-DEC-93      11.4      4.3
KEENE        21-MAR-93        4.4      -18
KEENE        22-JUN-93        29.5     19.3
KEENE        23-SEP-93        37.7     28.1
KEENE        22-DEC-93       -22      -18

insert into COMFORT values ('SAN FRANCISCO',
                           TO_DATE('21-MAR-1993','DD-MON-YYYY'),16.9,5.7,.5);
```

```

insert into COMFORT values ('SAN FRANCISCO',
    TO_DATE('22-JUN-1993','DD-MON-YYYY'),10.6,22.2,.1);
insert into COMFORT values ('SAN FRANCISCO',
    TO_DATE('23-SEP-1993','DD-MON-YYYY'),NULL,16.4,.1);
insert into COMFORT values ('SAN FRANCISCO',
    TO_DATE('22-DEC-1993','DD-MON-YYYY'),11.4,4.3,2.3);
insert into COMFORT values ('KEENE',
    TO_DATE('21-MAR-1993','DD-MON-YYYY'),4.4,-18,4.4);
insert into COMFORT values ('KEENE',
    TO_DATE('22-JUN-1993','DD-MON-YYYY'),29.5,19.3,1.3);
insert into COMFORT values ('KEENE',
    TO_DATE('23-SEP-1993','DD-MON-YYYY'),37.7,28.1,NULL);
insert into COMFORT values ('KEENE',
    TO_DATE('22-DEC-1993','DD-MON-YYYY'),-22,-18,3.9);

```

La tabella COMPITO

```

drop table COMPITO;
create table COMPITO (
    Compito      VARCHAR2(25) not null,
    Descrizione  VARCHAR2(80)
);

```

| | |
|-------------|--|
| Trebbiatore | Badare ai cavalli, condurli, regolare le lame |
| Aratore | Badare ai cavalli, condurli, spingere arare |
| Scavatore | Segnare e aprire il terreno, scavare, punteggiare, riempire, riassettaggere |
| Fabbro | Accendere il fuoco, usare il soffietto, tagliare, ferrare i cavalli |
| Taglialegna | Segnare e abbattere alberi, spaccare, accatastare, trasportare |
| Operaio | Lavoro generico senza compiti particolari |

```

insert into COMPITO values (
    'TAGLIALEGNA','SEGNARE E ABBATTERE ALBERI, SPACCARE, ACCATASTARE,  
TRASPORTARE');
insert into COMPITO values (
    'TREBBIATORE','BADARE AI CAVALLI, CONDURLI, REGOLARE LE LAME');
insert into COMPITO values (
    'FABBRO','ACCENDERE IL FUOCO, USARE IL SOFFIETTO, TAGLIARE, FERRARE I  
CAVALLI');
insert into COMPITO values (
    'SCAVATORE','SEGNARE E APRIRE IL TERRENO, SCAVARE, PUNTELLARE, RIEMPIRE,  
RIASSETTAGGIRE');
insert into COMPITO values (
    'ARATORE','BADARE AI CAVALLI, CONDURLI, SPINGERE, ARARE');
insert into COMPITO values (
    'OPERAIO','LAVORO GENERICO SENZA COMPITI PARTICOLARI');

```

La tabella COMPITOLAVORATORE

```
drop table COMPITOLAVORATORE;
create table COMPITOLAVORATORE (
Nome          VARCHAR2(25) not null,
Compito       VARCHAR2(25) not null,
Capacita      VARCHAR2(15)
);
;

insert into COMPITOLAVORATORE values ('DICK JONES','FABBRO','ECCELLENTE');
insert into COMPITOLAVORATORE values ('JOHN PEARSON','TREBBIATORE',null);
insert into COMPITOLAVORATORE values ('JOHN PEARSON','FABBRO','MEDIO');
insert into COMPITOLAVORATORE values ('HELEN BRANDT','TREBBIATORE',
'MOLTO VELOCE');
insert into COMPITOLAVORATORE values ('JOHN PEARSON','TAGLIALEGNA','BUONO');
insert into COMPITOLAVORATORE values ('VICTORIA LYNN','FABBRO','PRECISO');
insert into COMPITOLAVORATORE values ('ADAH TALBOT','OPERAIO','BUONO');
insert into COMPITOLAVORATORE values ('WILFRED LOWELL','OPERAIO','MEDIO');
insert into COMPITOLAVORATORE values ('ELBERT TALBOT','ARATORE','LENTO');
insert into COMPITOLAVORATORE values ('WILFRED LOWELL','ARATORE','MEDIO');
```

La tabella COMPLEANNO

```
drop table COMPLEANNO;
create table COMPLEANNO (
Nome          VARCHAR2(15),
Cognome       VARCHAR2(15),
DataNascita   DATE,
Eta           NUMBER
);

insert into COMPLEANNO values ('GEORGE','SAND',
TO_DATE('12-MAY-1946','DD-MON-YYYY'),42);
insert into COMPLEANNO values ('ROBERT','JAMES',
TO_DATE('23-AUG-1937','DD-MON-YYYY'),52);
insert into COMPLEANNO values ('NANCY','LEE',
TO_DATE('02-FEB-1947','DD-MON-YYYY'),42);
insert into COMPLEANNO values ('VICTORIA','LYNN',
TO_DATE('20-MAY-1949 3:27','DD-MON-YYYY HH24:MI'),42);
insert into COMPLEANNO values ('FRANK','PILOT',
TO_DATE('11-NOV-1942','DD-MON-YYYY'),42);
```

La tabella CONSEGNA

```
drop table CONSEGNA;
create table CONSEGNA (
```

```

Cliente      VARCHAR2(13),
Peso         NUMBER
);

insert into CONSEGNA values ('JOHNSON TOOL',59);
insert into CONSEGNA values ('DAGG SOFTWARE',27);
insert into CONSEGNA values ('TULLY ANDOVER',NULL);

```

La tabella DIPENDENTE

```

drop table DIPENDENTE;
create table DIPENDENTE(
Nome        VARCHAR2(25) not null,
Alloggio    VARCHAR2(15),
Eta         NUMBER
);

insert into DIPENDENTE values ('ADAH TALBOT', 'PAPA KING', 23);
insert into DIPENDENTE values ('DICK JONES', 'ROSE HILL', 18);
insert into DIPENDENTE values ('DONALD ROLLO', 'MATT'S', 16);
insert into DIPENDENTE values ('ELBERT TALBOT', 'WEITBROCHT', 43);
insert into DIPENDENTE values ('GEORGE OSCAR', 'ROSE HILL', 41);
insert into DIPENDENTE values ('PAT LAVAY', 'ROSE HILL', 21);
insert into DIPENDENTE values ('PETER LAWSON', 'CRANMER', 25);
insert into DIPENDENTE values ('WILFRED LOWELL', null, 67);

```

La tabella DUENOMI

```

drop table DUENOMI;
create table DUENOMI (
PrimoNome   VARCHAR2(25),
Cognome     VARCHAR2(25)
);

```

La tabella FATTURA

```

drop table FATTURA;
create table FATTURA
(NomeCliente  VARCHAR2(25),
DataFattura   DATE,
Importo       NUMBER(9,2));
insert into FATTURA values
('ELBERT TALBOT', TO_DATE('23-OCT-1901','DD-MON-YYYY'),5.03);
insert into FATTURA values
('JOHN PEARSON',   TO_DATE('09-NOV-1901','DD-MON-YYYY'),2.02);
insert into FATTURA values
('DICK JONES',     TO_DATE('12-SEP-1901','DD-MON-YYYY'),11.12);

```

```
insert into FATTURA values
('GENERAL STORE', TO_DATE('09-NOV-1901','DD-MON-YYYY'),22.10);
insert into FATTURA values
('ADAH TALBOT', TO_DATE('17-NOV-1901','DD-MON-YYYY'),8.29);
insert into FATTURA values
('GENERAL STORE', TO_DATE('01-SEP-1901','DD-MON-YYYY'),21.32);
insert into FATTURA values
('ADAH TALBOT', TO_DATE('15-NOV-1901','DD-MON-YYYY'),7.33);
insert into FATTURA values
('GENERAL STORE', TO_DATE('04-OCT-1901','DD-MON-YYYY'),8.42);
insert into FATTURA values
('KAY WALLBOM', TO_DATE('04-OCT-1901','DD-MON-YYYY'),1.43);
insert into FATTURA values
('JOHN PEARSON', TO_DATE('13-OCT-1901','DD-MON-YYYY'),12.41);
insert into FATTURA values
('DICK JONES', TO_DATE('23-OCT-1901','DD-MON-YYYY'),4.49);
insert into FATTURA values
('GENERAL STORE', TO_DATE('23-NOV-1901','DD-MON-YYYY'),40.36);
insert into FATTURA values
('GENERAL STORE', TO_DATE('30-OCT-1901','DD-MON-YYYY'),7.47);
insert into FATTURA values
('MORRIS ARNOLD', TO_DATE('03-OCT-1901','DD-MON-YYYY'),3.55);
insert into FATTURA values
('ROLAND BRANDT', TO_DATE('22-OCT-1901','DD-MON-YYYY'),13.65);
insert into FATTURA values
('MORRIS ARNOLD', TO_DATE('21-SEP-1901','DD-MON-YYYY'),9.87);
insert into FATTURA values
(VICTORIA LYNN', TO_DATE('09-OCT-1901','DD-MON-YYYY'),8.98);
insert into FATTURA values
('GENERAL STORE', TO_DATE('22-OCT-1901','DD-MON-YYYY'),17.58);
```

La tabella FESTA

```
drop table FESTA;
create table FESTA (
FESTA          VARCHAR2(25),
DataEff        DATE,
DataCelebrata DATE
);
insert into FESTA values ('NEW YEAR DAY',
    TO_DATE('01-JAN-1995','DD-MON-YYYY'),
    TO_DATE('01-JAN-1995','DD-MON-YYYY'));
insert into FESTA values ('MARTIN LUTHER KING, JR.',
    TO_DATE('15-JAN-1995','DD-MON-YYYY'),
    TO_DATE('16-JAN-1995','DD-MON-YYYY'));
insert into FESTA values ('LINCOLNS COMPLEANNO',
    TO_DATE('12-FEB-1995','DD-MON-YYYY'),
    TO_DATE('20-FEB-1995','DD-MON-YYYY'));
REM insert into FESTA values ('VALENTINES DAY',
    TO_DATE('14-FEB-1995','DD-MON-YYYY'),
```

```

TO_DATE('14-FEB-1995','DD-MON-YYYY'));
insert into FESTA values ('WASHINGTONS COMPLEANNO',
    TO_DATE('22-FEB-1995','DD-MON-YYYY'),
    TO_DATE('20-FEB-1995','DD-MON-YYYY'));
insert into FESTA values ('FAST DAY, NEW HAMPSHIRE',
    TO_DATE('22-FEB-1995','DD-MON-YYYY'),
    TO_DATE('22-FEB-1995','DD-MON-YYYY'));
REM insert into FESTA values ('VICTORIA DAY,CANADA',
    TO_DATE('21-MAY-1995','DD-MON-YYYY'),
    TO_DATE('21-MAY-1995','DD-MON-YYYY'));
insert into FESTA values ('MEMORIAL DAY',
    TO_DATE('30-MAY-1995','DD-MON-YYYY'),
    TO_DATE('29-MAY-1995','DD-MON-YYYY'));
REM insert into FESTA values ('FLAG DAY',
    TO_DATE('14-JUN-1995','DD-MON-YYYY'),
    TO_DATE('14-JUN-1995','DD-MON-YYYY'));
REM insert into FESTA values ('FATHERS DAY',
    TO_DATE('18-JUN-1995','DD-MON-YYYY'),
    TO_DATE('18-JUN-1995','DD-MON-YYYY'));
insert into FESTA values ('INDEPENDENCE DAY',
    TO_DATE('04-JUL-1995','DD-MON-YYYY'),
    TO_DATE('04-JUL-1995','DD-MON-YYYY'));
insert into FESTA values ('LABOR DAY',
    TO_DATE('04-SEP-1995','DD-MON-YYYY'),
    TO_DATE('04-SEP-1995','DD-MON-YYYY'));
insert into FESTA values ('COLUMBUS DAY',
    TO_DATE('08-OCT-1995','DD-MON-YYYY'),
    TO_DATE('09-OCT-1995','DD-MON-YYYY'));
REM insert into FESTA values ('HALLOWEEN',
    TO_DATE('31-OCT-1995','DD-MON-YYYY'),
    TO_DATE('31-OCT-1995','DD-MON-YYYY'));
REM insert into FESTA values ('VETERANS DAY',
    TO_DATE('11-NOV-1995','DD-MON-YYYY'),
    TO_DATE('11-NOV-1995','DD-MON-YYYY'));
insert into FESTA values ('THANKSGIVING',
    TO_DATE('23-NOV-1995','DD-MON-YYYY'),
    TO_DATE('23-NOV-1995','DD-MON-YYYY'));
REM insert into FESTA values ('CHANUKAH',
    TO_DATE('18-DEC-1995','DD-MON-YYYY'),
    TO_DATE('18-DEC-1995','DD-MON-YYYY'));
REM insert into FESTA values ('CHRISTMAS',
    TO_DATE('25-DEC-1995','DD-MON-YYYY'),
    TO_DATE('25-DEC-1995','DD-MON-YYYY'));

```

La tabella GIORNALE

```

drop table GIORNALE;

create table GIORNALE (
Argomento      VARCHAR2(15) not null,

```

```
Sezione      CHAR(1),
Pagina       NUMBER
);

insert into GIORNALE values ('Notizie'      , 'A'      , 1);
insert into GIORNALE values ('Sport'        , 'D'      , 1);
insert into GIORNALE values ('Editoriali'   , 'A'      , 12);
insert into GIORNALE values ('Economia'    , 'E'      , 1);
insert into GIORNALE values ('Meteo'        , 'C'      , 2);
insert into GIORNALE values ('Televisione'  , 'B'      , 7);
insert into GIORNALE values ('Nascite'      , 'F'      , 7);
insert into GIORNALE values ('Annunci'      , 'F'      , 8);
insert into GIORNALE values ('Vita moderna', 'B'      , 1);
insert into GIORNALE values ('Fumetti'      , 'C'      , 4);
insert into GIORNALE values ('Film'         , 'B'      , 4);
insert into GIORNALE values ('Bridge'       , 'B'      , 2);
insert into GIORNALE values ('Necrologi'    , 'F'      , 6);
insert into GIORNALE values ('Salute'       , 'F'      , 6);
```

La tabella GIORNO

```
drop table GIORNO;
create table GIORNO (
Giorno        DATE
);

insert into GIORNO values (TO_DATE('15-DEC-1901','DD-MON-YYYY'));
```

La tabella GIORNOPAGA

```
drop table GIORNOPAGA;
create table GIORNOPAGA (
DataCiclo     DATE
);

insert into GIORNOPAGA values (TO_DATE('15-JAN-1995','DD-MON-YYYY'));
insert into GIORNOPAGA values (TO_DATE('15-FEB-1995','DD-MON-YYYY'));
insert into GIORNOPAGA values (TO_DATE('15-MAR-1995','DD-MON-YYYY'));
insert into GIORNOPAGA values (TO_DATE('15-APR-1995','DD-MON-YYYY'));
insert into GIORNOPAGA values (TO_DATE('15-MAY-1995','DD-MON-YYYY'));
insert into GIORNOPAGA values (TO_DATE('15-JUN-1995','DD-MON-YYYY'));
insert into GIORNOPAGA values (TO_DATE('15-JUL-1995','DD-MON-YYYY'));
insert into GIORNOPAGA values (TO_DATE('15-AUG-1995','DD-MON-YYYY'));
insert into GIORNOPAGA values (TO_DATE('15-SEP-1995','DD-MON-YYYY'));
insert into GIORNOPAGA values (TO_DATE('15-OCT-1995','DD-MON-YYYY'));
insert into GIORNOPAGA values (TO_DATE('15-NOV-1995','DD-MON-YYYY'));
insert into GIORNOPAGA values (TO_DATE('15-DEC-1995','DD-MON-YYYY'));
```

La tabella HOCKEY

```
drop table HOCKEY;
create table HOCKEY (
Squadra      VARCHAR2(20),
Vinta        NUMBER,
Persa        NUMBER,
Pari         NUMBER
);
insert into HOCKEY values ('Quebec'      , 6, 23, 4);
insert into HOCKEY values ('Detroit'     , 10, 18, 5);
insert into HOCKEY values ('Vancouver'   , 11, 16, 6);
insert into HOCKEY values ('NY Islanders', 11, 20, 4);
insert into HOCKEY values ('Washington'  , 13, 15, 4);
insert into HOCKEY values ('Pittsburgh'  , 13, 16, 3);
insert into HOCKEY values ('Calgary'    , 14, 11, 9);
insert into HOCKEY values ('St Louis'   , 14, 12, 6);
insert into HOCKEY values ('Winnipeg'   , 14, 13, 5);
insert into HOCKEY values ('NY Rangers' , 15, 14, 5);
insert into HOCKEY values ('New Jersey' , 15, 15, 3);
insert into HOCKEY values ('Edmonton'  , 16, 11, 7);
insert into HOCKEY values ('Philadelphia', 16, 14, 4);
insert into HOCKEY values ('Los Angeles', 16, 14, 3);
insert into HOCKEY values ('Hartford'   , 16, 17, 1);
insert into HOCKEY values ('Toronto'    , 16, 18, 0);
insert into HOCKEY values ('Boston'     , 17, 13, 3);
insert into HOCKEY values ('Minnesota' , 17, 15, 2);
insert into HOCKEY values ('Chicago'   , 19, 13, 2);
insert into HOCKEY values ('Montreal'  , 20, 13, 4);
insert into HOCKEY values ('Buffalo'   , 21, 9, 4);
```

La tabella INDIRIZZO

```
drop table INDIRIZZO;
create table INDIRIZZO (
Cognome      VARCHAR2(25),
Nome         VARCHAR2(25),
Via          VARCHAR2(50),
Citta        VARCHAR2(25),
Prov         CHAR(2),
Cap          NUMBER,
Telefono     VARCHAR2(12),
Pro          VARCHAR2(5));

insert into INDIRIZZO values (
'BAILEY',      'WILLIAM',
null,null,null,null,'213-293-0223',null);
insert into INDIRIZZO values (
'ADAMS',       'JACK',
null,null,null,null,'415-453-7530',null);
```

```
insert into INDIRIZZO values (
  'SEP',           'FELICIA',
  null,null,null,null,'214-522-8383',null);
insert into INDIRIZZO values (
  'DE MEDICI',    'LEFTY',
  null,null,null,null,'312-736-1166',null);
insert into INDIRIZZO values (
  'DEMIURGE',     'FRANK',
  null,null,null,null,'707-767-8900',null);
insert into INDIRIZZO values (
  'CASEY',         'WILLIS',
  null,null,null,null,'312-684-1414',null);
insert into INDIRIZZO values (
  'ZACK',          'JACK',
  null,null,null,null,'415-620-6842',null);
insert into INDIRIZZO values (
  'YARROW',        'MARY',
  null,null,null,949414302,'415-787-2178',null);
insert into INDIRIZZO values (
  'WERSCHKY',      'ARNY',
  null,null,null,null,'415-235-7387',null);
insert into INDIRIZZO values (
  'BRANT',         'GLEN',
  null,null,null,null,'415-526-7512',null);
insert into INDIRIZZO values (
  'EDGAR',         'THEODORE',
  null,null,null,null,'415-525-6252',null);
insert into INDIRIZZO values (
  'HARDIN',        'HUGGY',
  null,null,null,null,'617-566-0125',null);
insert into INDIRIZZO values (
  'HILD',          'PHIL',
  null,null,null,null,'603-934-2242',null);
insert into INDIRIZZO values (
  'LOEBEL',         'FRANK',
  null,null,null,null,'202-456-1414',null);
insert into INDIRIZZO values (
  'MOORE',          'MARY',
  null,null,null,601262460,'718-857-1638',null);
insert into INDIRIZZO values (
  'SZEPI',          'FELICIA',
  null,null,null,null,'214-522-8383',null);
insert into INDIRIZZO values (
  'ZIMMERMAN',      'FRED',
  null,null,null,null,'503-234-7491',null);
```

La tabella LAVORATORE

```
drop table LAVORATORE;
create table LAVORATORE (
```

```

Nome          VARCHAR2(25) not null,
Eta           NUMBER,
Alloggio     VARCHAR2(15)
)
;

insert into LAVORATORE values ('BART SARJEANT',22,'CRANMER');
insert into LAVORATORE values ('ELBERT TALBOT',43,'WEITBROCHT');
insert into LAVORATORE values ('DONALD ROLLO',16,'MATTES');
insert into LAVORATORE values ('JED HOPKINS',33,'MATTES');
insert into LAVORATORE values ('WILLIAM SWING',15,'CRANMER');
insert into LAVORATORE values ('JOHN PEARSON',27,'ROSE HILL');
insert into LAVORATORE values ('GEORGE OSCAR',41,'ROSE HILL');
insert into LAVORATORE values ('KAY AND PALMER WALLBOM',null,'ROSE HILL');
insert into LAVORATORE values ('PAT LAVAY',21,'ROSE HILL');
insert into LAVORATORE values ('RICHARD KOCH AND BROTHERS',null,'WEITBROCHT');
insert into LAVORATORE values ('DICK JONES',18,'ROSE HILL');
insert into LAVORATORE values ('ADAH TALBOT',23,'PAPA KING');
insert into LAVORATORE values ('ROLAND BRANDT',35,'MATTES');
insert into LAVORATORE values ('PETER LAWSON',25,'CRANMER');
insert into LAVORATORE values ('VICTORIA LYNN',32,'MULLERS');
insert into LAVORATORE values ('WILFRED LOWELL',67,null);
insert into LAVORATORE values ('HELEN BRANDT',15,null);
insert into LAVORATORE values ('GERHARDT KENTGEN',55,'PAPA KING');
insert into LAVORATORE values ('ANDREW DYE',29,'ROSE HILL');

```

La tabella LOCAZIONE

```

drop table LOCAZIONE;
create table LOCAZIONE (
Citta        VARCHAR2(25),
Paese        VARCHAR2(25),
Continente   VARCHAR2(25),
Latitudine   NUMBER,
NordSud      CHAR(1),
Longitudine  NUMBER,
EstOvest     CHAR(1));

insert into LOCAZIONE values (
'ATENE','GRECIA','EUROPA',37.58,'N',23.43,'E');
insert into LOCAZIONE values (
'CHICAGO','STATI UNITI','NORDAMERICA',41.53,'N',87.38,'W');
insert into LOCAZIONE values (
'CONAKRY','GUINEA','AFRICA',9.31,'N',13.43,'W');
insert into LOCAZIONE values (
'LIMA','PERU','SUDAMERICA',12.03,'S',77.03,'W');
insert into LOCAZIONE values (
'MADRAS','INDIA','INDIA',13.05,'N',80.17,'E');
insert into LOCAZIONE values (
'MANCHESTER','INGHILTERRA','EUROPA',53.30,'N',2.15,'W');

```

```
insert into LOCAZIONE values (
'MOSCA','RUSSIA','EUROPA',55.45,'N',37.35,'E');
insert into LOCAZIONE values (
'PARIGI','FRANCIA','EUROPA',48.52,'N',2.20,'E');
insert into LOCAZIONE values (
'SHENYANG','CINA','ASIA',41.48,'N',123.27,'E');
insert into LOCAZIONE values (
'ROMA','ITALIA','EUROPA',41.54,'N',12.29,'E');
insert into LOCAZIONE values (
'TOKYO','GIAPPONE','ASIA',35.42,'N',139.46,'E');
insert into LOCAZIONE values (
'SYDNEY','AUSTRALIA','AUSTRALIA',33.52,'S',151.13,'E');
insert into LOCAZIONE values (
'SPARTA','GRECIA','EUROPA',37.05,'N',22.27,'E');
insert into LOCAZIONE values (
'MADRID','SPAGNA','EUROPA',40.24,'N',3.41,'W');
```

La tabella MATEMATICA

```
drop table MATEMATICA;
create table MATEMATICA (
Nome          VARCHAR2(12),
Sopra         NUMBER,
Sotto         NUMBER,
Vuoto         NUMBER
);

insert into MATEMATICA values ('NUMERO INTERO',11,-22,null);
insert into MATEMATICA values ('DECIMALE INF',33.33,-44.44,null);
insert into MATEMATICA values ('DECIMALE MED',55.5,-55.5,null);
insert into MATEMATICA values ('DECIMALE SUP',66.666,-77.777,null);
```

La tabella NOME

```
drop table NOME;
create table NOME (
Nome          VARCHAR2(25)
);
insert into NOME values ('HORATIO NELSON');
insert into NOME values ('VALDO');
insert into NOME values ('MARIE DE MEDICIS');
insert into NOME values ('FLAVIUS JOSEPHUS');
insert into NOME values ('EDYTHE P. M. GAMMIERE');
```

La tabella PAGA

```
drop table PAGA;
create table PAGA (
```

```

Nome          VARCHAR2(25) not null,
PagaGiorno   NUMBER
);

insert into PAGA values ('ADAH TALBOT',1);
insert into PAGA values ('ANDREW DYE',.75);
insert into PAGA values ('BART SARJEANT',.75);
insert into PAGA values ('DICK JONES',1);
insert into PAGA values ('GEORGE OSCAR',1.25);
insert into PAGA values ('PAT LAVAY',1.25);

```

La tabella PASTORE

```

rem Richiede l'utilizzo di ORACLE8 Objects.
rem Richiede l'esistenza dei tipi ANIMALE_TY e ANIMALI_NT.

create table PASTORE
(NomePastore      VARCHAR2(25),
 Animali          ANIMALI_NT)
nested table ANIMALI store as ANIMALI_NT_TAB;

```

La tabella PRESTITO

```

rem Richiede l'utilizzo di ORACLE8 Objects.
rem Richiede che sia stato creato il tipo di dati ATTREZZI_VA.

create table PRESTITO
(Nome          VARCHAR2(25) primary key,
 Attrezzo     ATTREZZI_VA);

insert into PRESTITO values
('JED HOPKINS',
 ATTREZZI_VA('HAMMER','SLEDGE','AX'));

```

La tabella PROPOSTA

```

drop table PROPOSTA;
create table PROPOSTA
(Proposta_ID      NUMBER(10) primary key,
 Nome_Destinatario  VARCHAR2(25),
 Nome_Proposta    VARCHAR2(25),
 Breve_Descrizione VARCHAR2(1000),
 Proposta_Tpro     CLOB,
 Budget           BLOB,
 Lettera_Presentazione BFILE);

```

La tabella PROSPETTIVA

```
drop table PROSPETTIVA;
create table PROSPETTIVA (
Nome          VARCHAR2(25) not null,
Indirizzo     VARCHAR2(35)
);

insert into PROSPETTIVA values ('ADAH TALBOT','23 ZWING, EDMESTON');
insert into PROSPETTIVA values ('DORY KENSON','GEN. DEL., BAYBAC');
insert into PROSPETTIVA values ('ELBERT TALBOT','3 MILE ROAD, WALPOLE');
insert into PROSPETTIVA values ('GEORGE PHEPPS','206 POLE, KINGSLEY');
insert into PROSPETTIVA values ('PAT LAVAY','1 EASY ST, JACKSON');
insert into PROSPETTIVA values ('TED BUTCHER','RFD 1, BRIGHTON');
insert into PROSPETTIVA values ('JED HOPKINS','GEN. DEL., TURBOW');
insert into PROSPETTIVA values ('WILFRED LOWELL',NULL);
```

La tabella RADIUS_VALS

```
drop table RADIUS_VALS;
create table RADIUS_VALS
(Radius      NUMBER(5));
```

La tabella REGISTRO

```
drop table REGISTRO;
create table REGISTRO (
DataAzione    DATE,           /* quando */
Azione        VARCHAR2(8),   /* comprato, venduto, pagato, ricevut */
Articolo       VARCHAR2(30),  /* che cosa */
Quantita      NUMBER,        /* quanti */
TipoQuantita  VARCHAR2(10),  /* tipo di quantita */
Tasso         NUMBER,        /* quanto per tipo di quantita */
Importo       NUMBER(9,2),   /* Importo totale */
Persona        VARCHAR2(25)  /* A chi o da chi? */
);

insert into REGISTRO values (
TO_DATE('01-APR-1901','DD-MON-YYYY'),
'PAGATO','SEMINARE',1,'DAY',3,3,'RICHARD KOCH AND BROTHERS');
insert into REGISTRO values (
TO_DATE('02-MAY-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',1,1,'DICK JONES');
insert into REGISTRO values (
TO_DATE('03-JUN-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',1,1,'ELBERT TALBOT');
insert into REGISTRO values (
TO_DATE('04-JAN-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',1,1,'GERHARDT KENTGEN');
```

```
insert into REGISTRO values (
TO_DATE('04-FEB-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',.5,'DAY',1,.5,'ELBERT TALBOT');
insert into REGISTRO values (
TO_DATE('05-APR-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',1,1,'DICK JONES');
insert into REGISTRO values (
TO_DATE('06-AUG-1901','DD-MON-YYYY'),
'PAGATO','SEMINARE',1,'DAY',1.8,1.8,'VICTORIA LYNN');
insert into REGISTRO values (
TO_DATE('07-OCT-1901','DD-MON-YYYY'),
'PAGATO','SEMINARE',.5,'DAY',3,1.5,'RICHARD KOCH AND BROTHERS');
prompt Going into silent mode.
set termout off
insert into REGISTRO values (
TO_DATE('09-SEP-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',1,1,'ADAH TALBOT');
insert into REGISTRO values (
TO_DATE('09-OCT-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',.5,'DAY',1.25,.63,'DONALD ROLLO');
insert into REGISTRO values (
TO_DATE('10-NOV-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',1.25,1.25,'JOHN PEARSON');
insert into REGISTRO values (
TO_DATE('10-AUG-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',1,1,'HELEN BRANDT');
insert into REGISTRO values (
TO_DATE('11-AUG-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',2,2,'HELEN BRANDT');
insert into REGISTRO values (
TO_DATE('11-SEP-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',.75,.75,'ROLAND BRANDT');
insert into REGISTRO values (
TO_DATE('12-DEC-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',1,1,'BART SARJEANT');
insert into REGISTRO values (
TO_DATE('12-JAN-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',1,1,'GEORGE OSCAR');
insert into REGISTRO values (
TO_DATE('13-JUN-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',1,1,'PETER LAWSON');
insert into REGISTRO values (
TO_DATE('14-JUL-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',1.2,1.2,'WILFRED LOWELL');
insert into REGISTRO values (
TO_DATE('15-JUL-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',2.25,2.25,'KAY AND PALMER WALLBOM');
insert into REGISTRO values (
TO_DATE('03-OCT-1901','DD-MON-YYYY'),
'VENDUTO','STIVALI PER CAVALLI',1,'EACH',12.5,12.5,'GARY KENTGEN');
insert into REGISTRO values (
```

```
TO_DATE('01-NOV-1901','DD-MON-YYYY'),  
'COMPRATO','VITELLO',2,'EACH',2,4,'GARY KENTGEN');  
insert into REGISTRO values (  
TO_DATE('02-NOV-1901','DD-MON-YYYY'),  
'COMPRATO','CAVALLA',1,'EACH',5,5,'JAMES COLE');  
insert into REGISTRO values (  
TO_DATE('03-NOV-1901','DD-MON-YYYY'),  
'COMPRATO','MAIALE',1,'EACH',2,2,'ANDREW DYE');  
insert into REGISTRO values (  
TO_DATE('04-NOV-1901','DD-MON-YYYY'),  
'COMPRATO','FIENO',1,'WAGON',5,5,'ANDREW DYE');  
insert into REGISTRO values (  
TO_DATE('05-NOV-1901','DD-MON-YYYY'),  
'COMPRATO','FIENO',4,'WAGON',5,20,'ANDREW DYE');  
insert into REGISTRO values (  
TO_DATE('05-NOV-1901','DD-MON-YYYY'),  
'COMPRATO','LINO',1,'SET',.75,.75,'ANDREW DYE');  
insert into REGISTRO values (  
TO_DATE('06-NOV-1901','DD-MON-YYYY'),  
'COMPRATO','COLT',2,'COLT',4.5,9,'ANDREW DYE');  
insert into REGISTRO values (  
TO_DATE('06-AUG-1901','DD-MON-YYYY'),  
'PAGATO','SEMINARE',2,'DAY',2,4,'ANDREW DYE');  
insert into REGISTRO values (  
TO_DATE('07-NOV-1901','DD-MON-YYYY'),  
'PAGATO','LEGNA TAGLIATA',1,'DAY',.5,.5,'ANDREW DYE');  
insert into REGISTRO values (  
TO_DATE('09-NOV-1901','DD-MON-YYYY'),  
'COMPRATO','COLT',1,'EACH',10,10,'ANDREW DYE');  
insert into REGISTRO values (  
TO_DATE('10-NOV-1901','DD-MON-YYYY'),  
'VENDUTO','LATTE',1,'EACH',28,28,'PAT LAVAY');  
insert into REGISTRO values (  
TO_DATE('11-NOV-1901','DD-MON-YYYY'),  
'VENDUTO','STIVALI PER CAVALLI',1,'EACH',6,6,'PAT LAVAY');  
insert into REGISTRO values (  
TO_DATE('11-NOV-1901','DD-MON-YYYY'),  
'VENDUTO','BURRO',1,'LB',.15,.15,'PAT LAVAY');  
insert into REGISTRO values (  
TO_DATE('12-NOV-1901','DD-MON-YYYY'),  
'PAGATO','LAVORARE',2,'DAY',.75,1.5,'PAT LAVAY');  
insert into REGISTRO values (  
TO_DATE('13-NOV-1901','DD-MON-YYYY'),  
'PAGATO','TAGLIARE CEPPI',.5,'DAY',.5,.25,'PAT LAVAY');  
insert into REGISTRO values (  
TO_DATE('13-NOV-1901','DD-MON-YYYY'),  
'PAGATO','ESTRARRE CEPPI',1.5,'DAY',.5,.75,'PAT LAVAY');  
insert into REGISTRO values (  
TO_DATE('13-DEC-1901','DD-MON-YYYY'),  
'PAGATO','LEGNA TAGLIATA',1,'DAY',.5,.5,'PAT LAVAY');  
insert into REGISTRO values (  
TO_DATE('14-NOV-1901','DD-MON-YYYY'),
```

```
'VENDUTO','LATTE',1,'EACH',35,35,'MORRIS ARNOLD');  
insert into REGISTRO values (  
TO_DATE('15-NOV-1901','DD-MON-YYYY'),  
'VENDUTO','MANZO',37,'LB',.04,1.48,'FRED FULLER');  
insert into REGISTRO values (  
TO_DATE('16-NOV-1901','DD-MON-YYYY'),  
'VENDUTO','BURRO',5,'LB',.16,.8,'VICTORIA LYNN');  
insert into REGISTRO values (  
TO_DATE('18-NOV-1901','DD-MON-YYYY'),  
'VENDUTO','BURRO',6,'LB',.16,.96,'JOHN PEARSON');  
insert into REGISTRO values (  
TO_DATE('20-NOV-1901','DD-MON-YYYY'),  
'VENDUTO','LATTE',1,'EACH',30,30,'PALMER WALLBOM');  
insert into REGISTRO values (  
TO_DATE('21-NOV-1901','DD-MON-YYYY'),  
'VENDUTO','MANZO',116,'LB',.06,6.96,'ROLAND BRANDT');  
insert into REGISTRO values (  
TO_DATE('22-NOV-1901','DD-MON-YYYY'),  
'VENDUTO','MANZO',118,'LB',.06,7.08,'GERHARDT KENTGEN');  
insert into REGISTRO values (  
TO_DATE('01-DEC-1901','DD-MON-YYYY'),  
'COMPRATO','MANZO',138,'LB',.05,6.9,'VICTORIA LYNN');  
insert into REGISTRO values (  
TO_DATE('01-DEC-1901','DD-MON-YYYY'),  
'COMPRATO','MANZO',130,'LB',.06,7.8,'GEORGE B. MCCORMICK');  
insert into REGISTRO values (  
TO_DATE('03-DEC-1901','DD-MON-YYYY'),  
'COMPRATO','MANZO',130,'LB',.05,6.5,'PETER LAWSON');  
insert into REGISTRO values (  
TO_DATE('03-DEC-1901','DD-MON-YYYY'),  
'COMPRATO','MANZO',125,'LB',.06,7.5,'HELEN BRANDT');  
insert into REGISTRO values (  
TO_DATE('05-DEC-1901','DD-MON-YYYY'),  
'COMPRATO','MANZO',140,'LB',.05,7,'ROBERT JAMES');  
insert into REGISTRO values (  
TO_DATE('05-DEC-1901','DD-MON-YYYY'),  
'COMPRATO','MANZO',145,'LB',.05,7.25,'ISAIAH JAMES');  
insert into REGISTRO values (  
TO_DATE('07-DEC-1901','DD-MON-YYYY'),  
'COMPRATO','CAVALLO',1,'EACH',30,30,'GEORGE AUGUST');  
insert into REGISTRO values (  
TO_DATE('07-DEC-1901','DD-MON-YYYY'),  
'COMPRATO','MIETITRICE',1,'EACH',47.5,47.5,'JANICE TALBOT');  
insert into REGISTRO values (  
TO_DATE('03-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','PANE',1,'BUSHEL',1.25,1.25,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('09-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','GRISSINI',1,'BOX',.5,.5,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('11-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','ZUPPA RISO',1,'EACH',.15,.15,'GENERAL STORE');
```

```
insert into REGISTRO values (
TO_DATE('11-JAN-1901','DD-MON-YYYY',
'CÖMPRATO','FILATI',1,'BOX',.05,.05,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('13-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','PAGLIA',10,'EACH',.03,.3,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('14-JAN-1901','DD-MON-YYYY'),
'VÈNDUTO','MANZO',1,'EACH',5.46,5.46,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('14-JAN-1901','DD-MON-YYYY'),
'VÈNDUTO','FORMAGGIO',13,'EACH',3.15,40.95,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('14-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','LANTERNA',1,'EACH',.1,.1,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('15-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','FRANCOBOLLI PER LETTERA',1,'EACH',.02,.02,'POST OFFICE');
insert into REGISTRO values (
TO_DATE('15-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','LIQUORE',2,'PAIR',.15,.3,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('16-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','OLIO',4,'GALLON',.1,.4,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('16-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','ZUCCHERO',25,'LB',.07,1.75,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('16-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','MELASSA',1,'GALLON',.6,.6,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('16-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','CARTOLINE AUGURI',1,'EACH',.3,.3,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('17-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','SELLA CAVALLO',1,'EACH',.85,.85,'LIVERY');
insert into REGISTRO values (
TO_DATE('17-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','GRANO',230,'LB',.01,2.3,'FEED STORE');
insert into REGISTRO values (
TO_DATE('18-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','MAIS',213,'LB',.01,2.13,'FEED STORE');
insert into REGISTRO values (
TO_DATE('18-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','CARTA',50,'SHEETS',.01,.5,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('18-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','CAFFE',1,'LB',.3,.3,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('18-JAN-1901','DD-MON-YYYY'),
'CÖMPRATO','SEMI',1,'LB',.12,.12,'GENERAL STORE');
insert into REGISTRO values (
```

```
TO_DATE('18-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','COTONE',3,'PAIR',.08,.24,'GENERAL STORE');  
insert into REGISTRO values (TO_DATE('19-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','COTONE',3,'PAIR',.08,.24,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('19-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','SECCHIO',24,'EACH',.03,.72,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('19-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','TELEFONATA',1,'EACH',.15,.15,'TELEFONO COMPANY');  
insert into REGISTRO values (  
TO_DATE('19-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','TE',.5,'LB',.5,.25,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('19-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','PALA',1,'EACH',.1,.1,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('19-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','SALE',1,'TABLET',.08,.08,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('19-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','BUSTE',6,'EACH',.02,.12,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('19-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','ACQUA',2,'QUART',.37,.74,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('23-JAN-1901','DD-MON-YYYY'),  
'VENDUTO','LEGNO',1,'CORD',2,2,'METHODIST CHURCH');  
insert into REGISTRO values (  
TO_DATE('24-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','ISTRUZIONE',1,'TERM',1,1,'SCHOOL');  
insert into REGISTRO values (  
TO_DATE('24-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','SABBIA',186,'TERM',.01,1.86,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('28-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','TERRA',1,'EACH',.9,.9,'MILL');  
insert into REGISTRO values (  
TO_DATE('28-JAN-1901','DD-MON-YYYY'),  
'COMPRATO','POPCORN',5,'LB',.04,.2,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('02-FEB-1901','DD-MON-YYYY'),  
'COMPRATO','SOLFATO',5,'LB',.25,1.25,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('03-FEB-1901','DD-MON-YYYY'),  
'COMPRATO','OLIO',4,'GALLON',.13,.52,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('03-FEB-1901','DD-MON-YYYY'),  
'COMPRATO','SUCCO CAROTA',1,'BOTTLE',.75,.75,'GENERAL STORE');  
insert into REGISTRO values (  
TO_DATE('04-FEB-1901','DD-MON-YYYY'),  
'COMPRATO','SCARPE',1,'EACH',.5,.5,'BLACKSMITH');
```

```
insert into REGISTRO values (
TO_DATE('04-FEB-1901','DD-MON-YYYY'),
'CÖMPRATO','TERRA',1,'EACH',.47,.47,'MILL');
insert into REGISTRO values (
TO_DATE('05-FEB-1901','DD-MON-YYYY'),
'CÖMPRATO','SIGARETTE',1,'BOX',.25,.25,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('07-FEB-1901','DD-MON-YYYY'),
'CÖMPRATO','ASCIA',2,'SPOOLS',.05,.1,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('08-FEB-1901','DD-MON-YYYY'),
'CÖMPRATO','MAGLIETTA',2,'EACH',.5,1,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('10-FEB-1901','DD-MON-YYYY'),
'VENDUTO','BURRO',9,'LB',.25,2.25,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('18-FEB-1901','DD-MON-YYYY'),
'CÖMPRATO','MEDICINA CAVALLO',1,'ENVELOPE',.13,.13,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('18-FEB-1901','DD-MON-YYYY'),
'CÖMPRATO','PIPA RADICA',1,'EACH',.15,.15,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('18-FEB-1901','DD-MON-YYYY'),
'VENDUTO','VITELLO',1,'EACH',4,4,'LILY CARLSTROM');
insert into REGISTRO values (
TO_DATE('25-FEB-1901','DD-MON-YYYY'),
'VENDUTO','BURRO',21,'LB',.25,5.25,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('28-FEB-1901','DD-MON-YYYY'),
'CÖMPRATO','SUCCO CAROTA',1,'BOTTLE',.75,.75,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('28-FEB-1901','DD-MON-YYYY'),
'CÖMPRATO','LIQUORE',1,'BOX',.2,.2,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('28-FEB-1901','DD-MON-YYYY'),
'VENDUTO','BURRO',3,'LB',.25,.75,'HELEN BRANDT');
insert into REGISTRO values (
TO_DATE('01-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','MACINA',1,'EACH',.45,.45,'MILL');
insert into REGISTRO values (
TO_DATE('06-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','MEDICINE INDIGESTIONE',1,'BOTTLE',.4,.4,
'DR. CARLSTROM');
insert into REGISTRO values (
TO_DATE('06-JUN-1901','DD-MON-YYYY'),
'CÖMPRATO','POLVERE DA SPARO',1,'BOX',.9,.9,'MILL');
insert into REGISTRO values (
TO_DATE('06-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','PANTALONI',1,'PAIR',.75,.75,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('07-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','ZUPPA VERDURA',200,'LB',.01,2,'MILL');
```

```
insert into REGISTRO values (
TO_DATE('08-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','TABACCO DA MASTICARE',1,'BOX',.25,.25,'MILL');
insert into REGISTRO values (
TO_DATE('07-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','SCARPE',1,'EACH',.35,.35,'BLACKSMITH');
insert into REGISTRO values (
TO_DATE('07-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','CHIODI',1,'BOX',.05,.05,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('07-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','POSTA',1,'EACH',1,1,'POST OFFICE');
insert into REGISTRO values (
TO_DATE('10-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','MANICOTTO TUBO STUFA',2,'EACH',.5,1,'VERNA HARDWARE');
insert into REGISTRO values (
TO_DATE('13-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','TERMOMETRO',1,'EACH',.15,.15,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('14-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','LOTTO CIMITERO N. 80',1,'EACH',25,25,'METHODIST CHURCH');
insert into REGISTRO values (
TO_DATE('14-MAR-1901','DD-MON-YYYY'),
'PÄGATO','SCAVO',1,'EACH',3,3,'JED HOPKINS');
insert into REGISTRO values (
TO_DATE('16-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','MACINA',1,'EACH',.16,.16,'MILL');
insert into REGISTRO values (
TO_DATE('16-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','MACINA',1,'EACH',.16,.16,'MILL');
insert into REGISTRO values (
TO_DATE('23-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','FODERE VESTITI',2,'YARD',.27,.54,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('18-AUG-1901','DD-MON-YYYY'),
'CÖMPRATO','TERMOMETRO',1,'EACH',1,1,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('25-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','SCARPE PER SHIRLEY',1,'PAIR',2.5,2.5,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('27-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','CONTENITORI SCIROPPO',2,'DOZEN',1.07,2.14,'VERNA HARDWARE');
insert into REGISTRO values (
TO_DATE('22-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','CONTENITORI LATTE',2,'EACH',2.5,5,'VERNA HARDWARE');
insert into REGISTRO values (
TO_DATE('23-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','DOPPIO FILTRO',1,'EACH',.95,.95,'VERNA HARDWARE');
insert into REGISTRO values (
TO_DATE('25-JUN-1901','DD-MON-YYYY'),
'CÖMPRATO','FILTRO LATTE',1,'EACH',.25,.25,'VERNA HARDWARE');
```

```
insert into REGISTRO values (
TO_DATE('27-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','ZUPPA VERDURA',77,'LB',.01,.77,'MILL');
insert into REGISTRO values (
TO_DATE('28-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','GRANO',104,'LB',.01,1.04,'MILL');
insert into REGISTRO values (
TO_DATE('06-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','FUNERALE',1,'EACH',3.19,3.19,'UNDERWOOD BROS');
insert into REGISTRO values (
TO_DATE('30-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','PENNELLO',1,'EACH',.05,.05,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('30-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','SABBIA',5,'BUSHEL',.03,.15,'QUARRY');
insert into REGISTRO values (
TO_DATE('31-MAR-1901','DD-MON-YYYY'),
'VENDUTO','MELASSA',3,'GALLON',1,3,'HAROLD SCHOLE');
insert into REGISTRO values (
TO_DATE('28-MAR-1901','DD-MON-YYYY'),
'VENDUTO','MELASSA',1,'GALLON',1,1,'GERHARDT KENTGEN');
insert into REGISTRO values (
TO_DATE('30-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','RIPARAZIONE OROLOGIO',1,'EACH',.25,.25,'MANNER JEWELERS');
insert into REGISTRO values (
TO_DATE('04-APR-1901','DD-MON-YYYY'),
'VENDUTO','BURRO',9,'LB',.23,2.07,'HAROLD SCHOLE');
insert into REGISTRO values (
TO_DATE('05-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','SODA',1,'BOTTLE',.05,.05,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('05-MAR-1901','DD-MON-YYYY'),
'CÖMPRATO','TELEFONATA',1,'EACH',.2,.2,'PHONE COMPANY');
insert into REGISTRO values (
TO_DATE('06-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','GUANTI',1,'PAIR',.25,.25,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('06-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','SCARPE PER SHIRLEY',1,'PAIR',2,2,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('09-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','ARACHIDI',1,'BAG',.05,.05,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('11-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','CRUSCA',300,'LB',.01,3,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('15-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','SCARPE',2,'EACH',.3,.6,'BLACKSMITH');
insert into REGISTRO values (
TO_DATE('17-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','ZUPPA VERDURA',173,'LB',.01,1.73,'GENERAL STORE');
```

```
insert into REGISTRO values (
TO_DATE('17-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','CRUSCA',450,'LB',.01,4.5,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('17-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','CARNE VITELLO',110,'LB',.01,1.1,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('22-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','ZUPPA VERDURA',454,'LB',.01,4.54,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('22-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','CRUSCA',300,'LB',.01,3,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('22-APR-1901','DD-MON-YYYY'),
'VENDUTO','VITELLO',1,'EACH',1,1,'PAT LAVAY');
insert into REGISTRO values (
TO_DATE('25-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','CARNE VITELLO',100,'EACH',.01,1,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('27-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','LUCIDO SCARPE',1,'EACH',.5,.5,'BLACKSMITH');
insert into REGISTRO values (
TO_DATE('07-JUN-1901','DD-MON-YYYY'),
'RECEIVED','PULEDRO',1,'EACH',5,5,'SAM DYE');
insert into REGISTRO values (
TO_DATE('07-JUN-1901','DD-MON-YYYY'),
'RECEIVED','CURA PULEDRO',1,'EACH',4,4,'SAM DYE');
insert into REGISTRO values (
TO_DATE('17-JUN-1901','DD-MON-YYYY'),
'|CÖMPRATO','TASSE SCUOLA',1,'EACH',6.56,6.56,'SCHOOL');
insert into REGISTRO values (
TO_DATE('17-JUN-1901','DD-MON-YYYY'),
'RECEIVED','TREBBIARE',2,'DAY',1,2,'HENRY CHASE');
insert into REGISTRO values (
TO_DATE('18-JUN-1901','DD-MON-YYYY'),
'PÄGATO','TREBBIARE',.5,'DAY',1,.5,'WILLIAM SWING');
insert into REGISTRO values (
TO_DATE('18-JUN-1901','DD-MON-YYYY'),
'CÖMPRATO','PECORA',22,'EACH',.87,19.14,'BOOLE AND JONES');
insert into REGISTRO values (
TO_DATE('15-MAR-1901','DD-MON-YYYY'),
'VENDUTO','PATATE',5,'BUSHEL',.25,1.25,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('15-MAR-1901','DD-MON-YYYY'),
'VENDUTO','MUCCA',2,'EACH',33,66,'SAM DYE');
insert into REGISTRO values (
TO_DATE('15-MAR-1901','DD-MON-YYYY'),
'RECEIVED','STIVALI PER CAVALLI',1,'EACH',10,10,'ADAH TALBOT');
insert into REGISTRO values (
TO_DATE('18-MAR-1901','DD-MON-YYYY'),
'VENDUTO','CARRO',1,'EACH',5,5,'ADAH TALBOT');
```

```
insert into REGISTRO values (
TO_DATE('04-APR-1901','DD-MON-YYYY'),
'VENDUTO','FINIMENTI',1,'EACH',2,2,'ADAH TALBOT');
insert into REGISTRO values (
TO_DATE('16-APR-1901','DD-MON-YYYY'),
'VENDUTO','MUCCA',3,'EACH',30,90,'GEORGE B. MCCORMICK');
insert into REGISTRO values (
TO_DATE('09-JUN-1901','DD-MON-YYYY'),
'COMPRATO','USO PASCOLO',1,'EACH',10,10,'GEORGE B. MCCORMICK');
insert into REGISTRO values (
TO_DATE('28-JUN-1901','DD-MON-YYYY'),
'COMPRATO','PECORA E TORO',1,'LOT',97.88,97.88,'EDWARD JOHNSON');
insert into REGISTRO values (
TO_DATE('03-JUL-1901','DD-MON-YYYY'),
'VENDUTO','LATTE',1,'EACH',35,35,'SAM DYE');
insert into REGISTRO values (
TO_DATE('18-MAY-1901','DD-MON-YYYY'),
'COMPRATO','VARIE',180,'LB',.01,1.8,'DEAN FOREMAN');
insert into REGISTRO values (
TO_DATE('20-MAY-1901','DD-MON-YYYY'),
'COMPRATO','VARIE',450,'LB',.01,4.5,'GEORGE OSCAR');
insert into REGISTRO values (
TO_DATE('22-MAY-1901','DD-MON-YYYY'),
'COMPRATO','VARIE',640,'LB',.01,6.4,'EDYTHE GAMMIERE');
insert into REGISTRO values (
TO_DATE('23-MAY-1901','DD-MON-YYYY'),
'COMPRATO','VARIE',110,'LB',.01,1.1,'JOHN AUSTIN');
insert into REGISTRO values (
TO_DATE('28-MAY-1901','DD-MON-YYYY'),
'COMPRATO','PETTINE',1,'EACH',.07,.07,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('29-MAY-1901','DD-MON-YYYY'),
'COMPRATO','BOTTONI',1,'BOX',.1,.1,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('05-JUL-1901','DD-MON-YYYY'),
'COMPRATO','FAVE',6,'LB',.03,.18,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('29-MAY-1901','DD-MON-YYYY'),
'COMPRATO','SEDANO',3,'LB',.08,.24,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('29-MAY-1901','DD-MON-YYYY'),
'COMPRATO','FORMAGGIO',3,'LB',.09,.27,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('04-JUN-1901','DD-MON-YYYY'),
'COMPRATO','BIRRA',1,'BOTTLE',.2,.2,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('04-JUN-1901','DD-MON-YYYY'),
'COMPRATO','SCIROPPO TOSSE',1,'BOTTLE',.25,.25,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('26-JUN-1901','DD-MON-YYYY'),
'COMPRATO','STRINGHE SCARPE',2,'PAIR',.04,.08,'GENERAL STORE');
```

```
insert into REGISTRO values (
TO_DATE('26-JUN-1901','DD-MON-YYYY'),
'CÓMPRAZO','GANCI CHIUSURA',1,'BOX',.05,.05,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('26-JUN-1901','DD-MON-YYYY'),
'CÓMPRAZO','PENNELLO PICCOLO',1,'BOX',.1,.1,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('06-MAR-1901','DD-MON-YYYY'),
'VÉNDUTO','UOVA',14,'DOZEN',.12,1.68,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('06-MAR-1901','DD-MON-YYYY'),
'VÉNDUTO','GALLINE',12,'EACH',.5,6,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('15-APR-1901','DD-MON-YYYY'),
'VÉNDUTO','UOVA',13,'DOZEN',.1,1.3,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('27-APR-1901','DD-MON-YYYY'),
'PÁGATO','SEMINARE',1,'DAY',3,3,'RICHARD KOCH AND BROTHERS');
insert into REGISTRO values (
TO_DATE('16-APR-1901','DD-MON-YYYY'),
'PÁGATO','SEMINARE',1,'DAY',3,3,'RICHARD KOCH AND BROTHERS');
insert into REGISTRO values (
TO_DATE('17-DEC-1901','DD-MON-YYYY'),
'PÁGATO','TAGLIARE',1,'DAY',.75,.75,'DICK JONES');
insert into REGISTRO values (
TO_DATE('28-JUL-1901','DD-MON-YYYY'),
'PÁGATO','TAGLIARE',1,'DAY',.75,.75,'DICK JONES');
insert into REGISTRO values (
TO_DATE('18-AUG-1901','DD-MON-YYYY'),
'PÁGATO','SARCHIARE',1,'DAY',.9,.9,'ELBERT TALBOT');
insert into REGISTRO values (
TO_DATE('29-SEP-1901','DD-MON-YYYY'),
'PÁGATO','LAVORARE',1,'DAY',1,1,'GERHARDT KENTGEN');
insert into REGISTRO values (
TO_DATE('19-JAN-1901','DD-MON-YYYY'),
'PÁGATO','LAVORARE',1,'DAY',1,1,'GERHARDT KENTGEN');
insert into REGISTRO values (
TO_DATE('30-JAN-1901','DD-MON-YYYY'),
'PÁGATO','LAVORARE',.5,'DAY',1,.5,'ELBERT TALBOT');
insert into REGISTRO values (
TO_DATE('28-FEB-1901','DD-MON-YYYY'),
'PÁGATO','LAVORARE',1,'DAY',1,1,'ELBERT TALBOT');
insert into REGISTRO values (
TO_DATE('20-MAR-1901','DD-MON-YYYY'),
'PÁGATO','LAVORARE',1,'DAY',1,1,'DICK JONES');
insert into REGISTRO values (
TO_DATE('21-JUL-1901','DD-MON-YYYY'),
'PÁGATO','LAVORARE',1,'DAY',1,1,'VICTORIA LYNN');
insert into REGISTRO values (
TO_DATE('22-OCT-1901','DD-MON-YYYY'),
'PÁGATO','SEMINARE',1,'DAY',1.8,1.8,'DICK JONES');
```

```
insert into REGISTRO values (
TO_DATE('23-SEP-1901','DD-MON-YYYY'),
'PAGATO','ARARE',.5,'DAY',3,1.5,'RICHARD KOCH AND BROTHERS');
insert into REGISTRO values (
TO_DATE('22-AUG-1901','DD-MON-YYYY'),
'PAGATO','TAGLIARE',1,'DAY',1,1,'PETER LAWSON');
insert into REGISTRO values (
TO_DATE('23-AUG-1901','DD-MON-YYYY'),
'PAGATO','TAGLIARE',1,'DAY',1,1,'PETER LAWSON');
insert into REGISTRO values (
TO_DATE('24-MAY-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',1.2,1.2,'WILFRED LOWELL');
insert into REGISTRO values (
TO_DATE('11-MAY-1901','DD-MON-YYYY'),
'PAGATO','LAVORARE',1,'DAY',1.2,1.2,'WILFRED LOWELL');
insert into REGISTRO values (
TO_DATE('26-JUN-1901','DD-MON-YYYY'),
'PAGATO','DIPINGERE',1,'DAY',1.75,1.75,'KAY AND PALMER WALLBOM');
insert into REGISTRO values (
TO_DATE('02-JUL-1901','DD-MON-YYYY'),
'COPRATO','MATITE',220,'LB',.01,2.2,'EDYTHE GAMMIERE');
insert into REGISTRO values (
TO_DATE('03-JUL-1901','DD-MON-YYYY'),
'COPRATO','MAIALE',1,'EACH',3,3,'JOHN AUSTIN');
insert into REGISTRO values (
TO_DATE('08-JUL-1901','DD-MON-YYYY'),
'COPRATO','FORMAGGIO',1,'LB',.09,.09,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('09-JUL-1901','DD-MON-YYYY'),
'COPRATO','BIRRA',1,'BOTTLE',.2,.2,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('02-AUG-1901','DD-MON-YYYY'),
'COPRATO','LATTE',3,'EACH',2.5,7.5,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('05-AUG-1901','DD-MON-YYYY'),
'COPRATO','ZUPPA VERDURA',120,'LB',.01,1.2,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('08-AUG-1901','DD-MON-YYYY'),
'COPRATO','PENNELLO',1,'EACH',.06,.06,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('12-AUG-1901','DD-MON-YYYY'),
'COPRATO','GRANO',90,'LB',.01,.9,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('25-MAR-1901','DD-MON-YYYY'),
'VENDUTO','MELASSA',5,'GALLON',1,5,'SAM DYE');
insert into REGISTRO values (
TO_DATE('29-AUG-1901','DD-MON-YYYY'),
'VENDUTO','BURRO',5,'LB',.23,1.15,'GERHARDT KENTGEN');
insert into REGISTRO values (
TO_DATE('06-SEP-1901','DD-MON-YYYY'),
'COPRATO','TELEFONATA',1,'EACH',.2,.2,'TELEFONO COMPANY');
```

```
insert into REGISTRO values (
TO_DATE('09-SEP-1901','DD-MON-YYYY'),
'CÖMPRATO','CARAMELLE',1,'BAG',.05,.05,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('12-SEP-1901','DD-MON-YYYY'),
'CÖMPRATO','CRUSCA',170,'LB',.01,1.7,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('13-SEP-1901','DD-MON-YYYY'),
'CÖMPRATO','SCARPE',4,'EACH',.3,1.2,'BLACKSMITH');
insert into REGISTRO values (
TO_DATE('15-SEP-1901','DD-MON-YYYY'),
'CÖMPRATO','ZUPPA VERDURA',144,'LB',.01,1.44,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('20-APR-1901','DD-MON-YYYY'),
'CÖMPRATO','CRUSCA',370,'LB',.01,3.7,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('17-JUL-1901','DD-MON-YYYY'),
'CÖMPRATO','VITELLO',90,'LB',.01,.9,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('20-JUL-1901','DD-MON-YYYY'),
'CÖMPRATO','ZUPPA VERDURA',300,'LB',.01,3,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('25-JUL-1901','DD-MON-YYYY'),
'VENDUTO','VITELLO',1,'EACH',1,1,'SAM DYE');
insert into REGISTRO values (
TO_DATE('19-SEP-1901','DD-MON-YYYY'),
'CÖMPRATO','CRUSCA',100,'LB',.01,1,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('23-SEP-1901','DD-MON-YYYY'),
'CÖMPRATO','VITELLO',110,'LB',.01,1.1,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('25-SEP-1901','DD-MON-YYYY'),
'CÖMPRATO','ZUPPA',80,'LB',.01,.8,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('07-OCT-1901','DD-MON-YYYY'),
'PÄGATO','LAVORARE',1,'DAY',1,1,'JED HOPKINS');
set termout on
prompt Waking up again.
insert into REGISTRO values (
TO_DATE('12-OCT-1901','DD-MON-YYYY'),
'CÖMPRATO','PECORA',12,'EACH',.9,10.8,'BOOLE AND JONES');
insert into REGISTRO values (
TO_DATE('15-OCT-1901','DD-MON-YYYY'),
'VENDUTO','MANZO',935,'LB',.03,28.05,'GENERAL STORE');
insert into REGISTRO values (
TO_DATE('18-OCT-1901','DD-MON-YYYY'),
'RECEIVED','STIVALI PER CAVALLI',1,'EACH',10,10,'ADAH TALBOT');
insert into REGISTRO values (
TO_DATE('12-OCT-1901','DD-MON-YYYY'),
'VENDUTO','LATTE',1,'EACH',35,35,'GEORGE B. MCCORMICK');
```

La tabella RIVISTA

```
drop table RIVISTA;
create table RIVISTA (
Nome      VARCHAR2(16),
Titolo    VARCHAR2(37),
Autore    VARCHAR2(25),
DataUscita DATE,
Pagina    NUMBER);

insert into RIVISTA values (
'BERTRAND MONTHLY','THE BARBERS WHO SHAVE THEMSELVES.',
'BONHOEFFER, DIETRICH',
TO_DATE('23-MAY-1988','DD-MON-YYYY'),70);
insert into RIVISTA values (
'LIVE FREE OR DIE',"HUNTING THOREAU IN NEW HAMPSHIRE",
'CHESTERTON, G.K.',
TO_DATE('26-AUG-1981','DD-MON-YYYY'),320);
insert into RIVISTA values (
'PSYCHOLOGICA','THE ETHNIC NEIGHBORHOOD',
'RUTH, GEORGE HERMAN',
TO_DATE('18-SEP-1919','DD-MON-YYYY'),246);
insert into RIVISTA values (
'FADED ISSUES','RELATIONAL DESIGN AND ENTHALPY',
'WHITEHEAD, ALFRED',
TO_DATE('20-JUN-1926','DD-MON-YYYY'),279);
insert into RIVISTA values (
'ENTROPY WIT',"INTERCONTINENTAL RELATIONS.",
'CROOKES, WILLIAM',
TO_DATE('20-SEP-1950','DD-MON-YYYY'),20);
```

La tabella ROSE

```
drop table ROSE;
create table ROSE (
Alloggio    VARCHAR2(12)
);

insert into ROSE values ('ROSELYN');
insert into ROSE values ('ROSE HILL');
insert into ROSE values ('ROSE GARDEN');
insert into ROSE values ('ROSE');
```

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
);
insert into STRANO values ('PLEASANT LAKE',
TO_DATE('21-MAR-1993','DD-MON-YYYY'), 39.99, -1.31, 3.6);
insert into STRANO values ('PLEASANT LAKE',
TO_DATE('22-JUN-1993','DD-MON-YYYY'), 101.44, 86.2, 1.63);
insert into STRANO values ('PLEASANT LAKE',
TO_DATE('23-SEP-1993','DD-MON-YYYY'), 92.85, 79.6, 1.00003);
insert into STRANO values ('PLEASANT LAKE',
TO_DATE('22-DEC-1993','DD-MON-YYYY'), -17.445, -10.4, 2.4);

```

La tabella TESTNUMERO

```

drop table TESTNUMERO;
create table TESTNUMERO (
Valore1        NUMBER,
Valore2        NUMBER,
Valore3        NUMBER(10,2)
);
insert into TESTNUMERO values (0,0,0);
insert into TESTNUMERO values (.0001,.0001,.0001);
insert into TESTNUMERO values (1234,1234,1234);
insert into TESTNUMERO values (1234.5,1234.5,1234.5);
insert into TESTNUMERO values (null,null,null);
insert into TESTNUMERO values (1234.56,1234.56,1234.56);
insert into TESTNUMERO values (1234.567,1234.567,1234.567);
insert into TESTNUMERO values
(98761234.567,98761234.567,98761234.567);

```

La tabella VIRGOLA

```

drop table VIRGOLA;
create table VIRGOLA (
Importo        NUMBER,
CarImporto     VARCHAR2(20)
);
insert into VIRGOLA values (0,'0');
insert into VIRGOLA values (0.25,'0.25');
insert into VIRGOLA values (1.25,'1.25');
insert into VIRGOLA values (12.25,'12.25');
insert into VIRGOLA values (123.25,'123.25');
insert into VIRGOLA values (1234.25,'1,234.25');
insert into VIRGOLA values (12345.25,'12,345.25');
insert into VIRGOLA values (123456.25,'123,456.25');
insert into VIRGOLA values (1234567.25,'1,234,567.25');
insert into VIRGOLA values (12345678.25,'12,345,678.25');
insert into VIRGOLA values (123456789.25,'123,456,789.25');

```

```
REM insert into VIRGOLA values (1234567890.25,'1,234,567,890.25');
REM insert into VIRGOLA values (12345678901.25,'12,345,678,901.25');
```

La tabella VUOTA

```
drop table VUOTA;
create table VUOTA (
Niente      VARCHAR2(25),
Meno        NUMBER
);
```

REMARK Nessun record. Questa tabella è vuota.

Indice analitico

Simboli

-- (due trattini), in SQLPLUS, 88, 100
- (meno), simbolo, per l'operatore MINUS, 561, 568
- (sottrazione), operatore, 140
- (trattino), simbolo, nel comando copy, 415
! (punto esclamativo), simbolo, 752
 nelle ricerche SOUNDEX di ConText, 566, 568
!= (diverso), operatore, 49-50
(cancelletto), simbolo, 276, 753
\$ (dollaro), simbolo, 397, 753
 per l'espansione etimologica di ConText, 565, 568
% (percentuale), simbolo, 51, 135, 753
 carattere jolly, 564, 568
%FOUND, attributo di cursore, 427-428, 754
%ISOPEN, attributo di cursore, 427-428, 755
%NOTFOUND, attributo di cursore, 427-428, 755
%ROWCOUNT, attributo di cursore, 427-428, 755
%ROWTYPE, attributo di cursore, 422, 756
%TYPE, attributo di cursore, 422, 757
& (e commerciale), simbolo
 per l'operatore AND, 560, 568
 prefisso ai nomi di variabili, 273
 inizio di variabili in SQLPLUS, 314
& o && (e commerciale o e commerciale doppia), simbolo, 758
() (parentesi), simbolo, 759
 con le funzioni, 121-123
 nelle ricerche di ConText, 568
* (asterisco), simbolo, 59
 in select, 320
 nelle ricerche tramite ConText, 568

* (moltiplicazione), simbolo, 759
 funzione, 140
** (esponenziale), 759
*/ (asterisco barra), simbolo, 100
, (virgola), simbolo
 per l'operatore ACCUM, 561, 568
 separatore di colonne, 332
. (punto [forma 1]), 760
. (punto [forma 2]), 760
. (punto), simbolo, 314
 fine dei blocchi PL/SQL, 420
. (a), 761
/ (barra [forma 2]), 761
/ (barra), simbolo
 esecuzione di comandi SQL, 279
 in SQLPLUS, 102-104
 nei blocchi PL/SQL, 420
 nelle date, 174
/ (divisione [forma 1]), 761
 funzione, 140
/* (barra asterisco), simbolo, 100
/* */ (commento), 761
 in SQLPLUS, 88
/*+...*/ , simbolo di hint, 288
/,-:, formato delle date, 178
:(due punti), simbolo, 45
 nel comando create trigger, 442
 nelle ricerche di ConText, 569
 prefisso di variabile host, 762
:= (uguale a), 762
;(punto e virgola), 762
 per l'operatore NEAR, 563, 568
 simbolo;terminatore SQL, 104, 279
? (punto interrogativo), simbolo, 753
 nelle ricerche fuzzy di ConText, 565, 568
@ (chiocciola [forma 1]), 763

- @ (chiocciola [forma 2]), 763
@ (chiocciola), simbolo, 405
@@ (doppia chiocciola), 763
[] (parentesi quadre), opzioni di funzioni, 142
^= (diverso), operatore, 50
_ (sottolineatura), simbolo, 51, 751
 carattere jolly, 564, 568
_a, suffisso, nei ruoli, 366
_ad, suffisso, nei ruoli, 366
_EDITOR, 752
{} (parentesi graffe), 763
 nelle ricerche di ConText, 563, 568
 nei grafici di SQLPLUS, 298
| (barra verticale), simbolo
 in SQLPLUS, 93
 per l'operatore OR, 561, 568
|| (doppia barra), funzione, 115
|| (doppia barra), simbolo di concatenazione, 314
| (barra verticale spezzata), 764
|| (concatenazione), 764
' (apice singolo), simbolo, 93, 758
" (doppi apici), simbolo, 752
+ (addizione), 759
 funzione, 140
+, segno, nelle unioni esterne, 225
< (minore), operatore, 49
<<>> (delimitatore di nome dell'etichetta PL/SQL), 763
<= (minore o uguale a), operatore, 50
<> (diverso da), operatore, 50
= (uguale a), operatore, 48
=> (uguale o maggiore di), simbolo del package CTX_DDL, 578
> (maggiore di), operatore, 48-49
> (maggiore di), simbolo, operatore di soglia in ConText, 559
>= (maggiore o uguale), operatore, 50
A.D., formato delle date, 178
A.M., formato delle date, 178
Abbreviazioni, 20-22
ABS (valore assoluto), funzione, 140, 145, 764
Accelerazione del processo produttivo, 18
ACCEPT, comando, 273, 310-311, 765
Accesso;a dati remoti, 403-418
 restrizioni, 412
Accodamento, 876
Account
 creazione, 354
 schemi, 458
ACCUM, operatore di ConText, 561, 568
ACCUMULATE, operatore, 765
ACOS, comando, 765
activity.sql, file di avvio, 106
ADD, clausola, nel comando alter table, 339
ADD, procedura del package
 DBMS_REFRESH, 545
ADD_MONTHS, funzione, 166-167, 766
Addizione, 143, 759
ADDRESS (ROW), 766
AFTER, parola chiave del comando create trigger, 441
AFTER, trigger, 439
AFTER ROW, trigger, 549
Alberi genealogici, 246-255
 esclusione di elementi e rami, 250-252
 spostamento verso le radici, 252-254
- ## A
- Alias, 766
 di colonne, 95, 207-208
 di viste, 208
ALL
 clausola, 157
 operatore, 766
ALL_CATALOG, vista, 612
ALL_CLU_COLUMNS, vista, 631
ALL_COL_COMMENTS, vista, 627
ALL_CONS_COLUMNS, vista, 624
ALL_CONSTRAINTS, vista, 623
ALL_DB_LINKS, vista, 636
ALL_DIRECTORY, vista, 635
ALL_ERRORS, vista, 641
ALL_IND_COLUMNS, vista, 630
ALL_INDEXES, vista, 629
ALL_OBJECTS, vista, 614
ALL_SEQUENCE, vista, 621
ALL_SNAPSHOTS, vista, 549
ALL_SNAPSHOTS, vista, 638
ALL_SOURCE, vista, 467
ALL_SOURCE, vista, 641
ALL_SYNONYMS, vista, 621
ALL_TAB_COLUMNS, vista, 617
ALL_TAB_COLUMNS, vista, 80
ALL_TAB_COMMENTS, vista, 626
ALL_TABLES, vista, 615
ALL_TRIGGERS, vista, 640
ALL_TYPE_ATTRS, vista, 80
ALL_VIEWS, vista, 620
ALLOCATE, comando, 769
ALTER ANY SNAPSHOT, privilegio, 552
ALTER ANY TABLE, privilegio, 448, 552

- ALTER ANY TRIGGER, privilegio, 448
 ALTER CLUSTER, comando, 769
 ALTER DATABASE, comando, 770
 ALTER FUNCTION, comando, 468, 773
 ALTER INDEX, comando, 380, 773
 ALTER PACKAGE, comando, 468, 774
 ALTER PROCEDURE, comando, 468, 775
 ALTER PROFILE, comando, 775
 ALTER RESOURCE COST, comando, 775
 ALTER ROLE, comando, 365-366, 776
 ALTER ROLLBACK SEGMENT, comando, 776
 ALTER SEQUENCE, comando, 776
 ALTER SESSION, comando, 311, 575, 777
 ALTER SESSION, comando, 777
 ALTER SNAPSHOT LOG, comando, 552, 779
 ALTER SYSTEM, comando, 780
 ALTER TABLE, comando, 332, 339, 350, 448, 781
 ALTER TABLESPACE, comando, 383, 784
 ALTER TRIGGER, comando, 448, 785
 ALTER TRIGGER COMPILE, comando, 449
 ALTER TYPE, comando, 486, 786
 ALTER USER, comando, 366-367, 786
 ALTER VIEW, comando, 787
 Ambienti grafici, 20
 Amministratore del database, 356
 Analisi, orientamento agli oggetti, 84-85
 ANALYZE, comando, 787
 AND, operatore booleano, 54, 788
 nelle ricerche tramite ConText, 560, 568
 combinazione con OR, 56-57
 Annidamento, 788
 Anno 2000, 184-185
 Annotazioni redo log, 345
 ANSI, 788
 ANY, operatore, 788
 API, 511
 APPEND
 comando, 104, 792
 hint con il comando insert, 287-288
 opzione del comando copy, 415
 procedura di DBMS_LOB, 515, 526-527
 Applicazioni, 792
 fasi di sviluppo, 23-24
 orientate alla lingua, 23
 progettazione, nuova visione, 19
 sviluppo, 19
 Approccio cooperativo, 5-6
 ARCH, PROCESSO, 793
 ARCHIVE_LOG, comando, 792
 Archiviare, 793
 Area di contesto, 793
 Argomento, 793
 Array, elaborazione, 793
 Array variabili, 74, 489, 793
 creazione, 490
 inserimento di record, 492-493
 problemi, 503-506
 selezione di dati, 494-495
 ARRAYSIZE, parametro, 414, 793
 Arrotondamenti negli inserimenti, 334-336
 AS
 clausola, 95, 207-208
 delimitatore, 793
 AS OBJECT, clausola, 77
 AS VARRAY, clausola del comando create type, 490
 ASCII
 formato, 105
 funzione, 794
 ASIN, comando, 794
 Astratti, tipi di dati, 72-73, 795
 Astrazione, concetto, 75
 ATAN, comando, 795
 ATAN2, comando, 795
 Attributi, 8, 83, 795
 AUDIT (forma 1, oggetti di schema), 795
 AUDIT (forma 2, istruzioni SQL), 797
 Auditing, 797-798
 CLUSTER, 797
 coda, 813
 DATABASE LINK, 797
 DIRECTORY, 797
 EXISTS, 797
 INDEX, 797
 NOT EXISTS, 797
 PROCEDURE, 797
 PROFILE, 797
 PUBLIC DATABASE LINK, 797
 PUBLIC SYNONYM, 797
 ROLE, 797
 ROLLBACK SEGMENT, 797
 SEQUENCE, 797
 SESSION, 797
 SYNONYM, 797
 SYSTEM AUDIT, 797
 SYSTEM GRANT, 797
 TABLE, 797
 TABLESPACE, 797
 TRIGGER, 797
 TYPE, 797

Auditing (*continua*)

 USER, 797

 VIEW, 797

AUTOCOMMIT, comando, 288, 798

Autorità, 353-370

Autorizzazione, 799

AVG, funzione, 140, 150, 799

Avvio, 799

AZIONE, tabella, 375

B

B.C., formato delle date, 178

Background, processo, 799

Backup in linea, 799

BEFORE, parola chiave del comando create trigger, 441

BEFORE, trigger, 439

BEGIN, istruzione, 423, 799

BETWEEN, operatore, 54, 61, 800

BETWEEN...AND, operatore, 55

BFILE, funzioni e procedure;utilizzo, 530

BFILE, tipo di dati, 74, 114, 507-508, 800

BFILENAME, procedura, 512

Binari, dati, formattati e RAW, 527

Binding, fase, 800

BITMAP

 clausola del comando index, 376
 conversione automatica in RowID, 376
 indici, 375-376, 800

Blaster, filtro, 577-578

BLOB, tipo di dati, 114, 507-508, 800

Blocchi di PL/SQL, 419-420

 sezione comandi, 423-424

 sezione dichiarazioni, 420-423

Blocchi sulle definizioni di dati, 800

Blocco, 801

 struttura, 801

Blocco (lock), 801

 a livello di riga, 802

 condiviso, 802

 di intestazione di segmento, 802

 di record, 802

 esclusivo, 802

 fisico, 802

 in modalità condivisa, 803

 modalità esclusiva, 802

 per aggiornamento condiviso, 803

BODY, clausola del comando drop package, 469

BREAK, 803

BREAK ON, comando, in combinazione con compute, 88, 96, 259-262, 267-270, 321

BREAK ON PAGE, comando, 268

BREAK ON REPORT, comando, 268

BREAK ON ROW, comando, 262, 268

BTITLE, comando, 87, 93, 262-264

BTITLE OFF, comando, 110

B-tree, formato, 374, 799

Buffer, 804

 corrente, 825

 database, 805

 di default, 278

 esecuzione, 279

 redo log, 805

 salvataggio di comandi SQLPLUS, 278

SQL, 101

C

C, linguaggio, 805

Cache, 805

 gestore, 805

Calcoli

 di totali, 320

 in if, then, else, 327-329

Campo, 805

Cancellazione di caratteri, 302

Caratteri, 114

 alfanumerici, 114

 cancellazione nelle stringhe, 309

 conversione maiuscoli e minuscoli, 124-125

 conversioni, 300-301

 eliminazione, 123, 302

 funzioni, 805

 non visualizzabili, 807

 sostituzioni, 196

Caratteri jolly, 51

 % (percentuale), 135

 nelle ricerche di ConText, 564

CASCADE, opzione del comando drop user, 357

CAST, parola chiave, 502

Casting, 502

CAT, sinonimo, 612

Cataloghi degli utenti, 612

Categoria, 807

 di comandi, 571

CEIL, funzione, 140, 145, 808

CHANGE

 comando, 102-104, 808

 procedura del package

 DBMS_REFRESH, 546

CHAR, tipo di dati, 114, 187, 809

 con SUBSTR, 129

- CHARTOROWID, funzione, 189, 195, 809
 CHECK, vincolo, 350
 Checkpoint, 809
 Chiamate ricorsive, 809
 Chiavi, 810
 - composte, 810
 - di cluster, 385
 - di testo, 575
 - intelligenti, 690
 - candidate, 336
 - esterne, 30, 337, 686-687, 810
 - primarie, 26, 85, 337, 375, 910
 - creazione, 379
 - e snapshot, 538-539
 - complete, 28
 - uniche, 810
 CHR, funzione, 810
 CHR(12), espressione, 325
 CHUNK, parametro, 510
 Cicli, 810
 - a cursore, 427-429, 811
 - FOR, 429-431
 - FOR a cursore, 431-432
 - nei blocchi PL/SQL, 426-433
 - semplici, 426-429
 - WHILE, 432-433
 Cicli di collaudo, 19
 Classi di dati, 75
 - ereditarietà, 76
 Clausole, 811
 - relazionali, 47
 CLEAR, comando, 811
 CLEAR BREAKS, comando, 110, 267
 CLEAR BUFFER, comando, 103
 CLEAR COLUMNS, comando, 110
 CLEAR COMPUTES, comando, 110, 267
 Client, 415-416, 812
 CLIENTE, tabella, 85
 CLIMA, tabella, 11
 CLOB, tipo di dati, 74, 114, 507-508, 812
 - scrittura di dati alfanumerici, 525
 CLOSE, istruzione, 424, 812
 CLU, sinonimo, 630
 Cluster, 385-386, 812
 - chiave, 812
 - colonne, 813
 - di hash, 386, 813
 - indice, 378, 813
 - informazioni tramite le viste, 630-631
 CLUSTER, clausola del comando create table, 385
 CMDSEP, comando, 813
 COALESCE, opzione, 813
 Coda di auditing, 813
 Codice
 - di PL/SQL, 419
 - orientato agli oggetti, 70-71
 Codici, 20-22
 - e lingua naturale, 22-23
 - generazione, 389-395
 - visualizzazione, 467
 Coerenza, 813
 - di lettura, 813
 Collegamenti dinamici, 413-415
 Collettori, 489
 - variabilità, 505
 Colonne, 321, 814
 - aggiunta, 339, 341
 - alias, 95, 207-208
 - attributo, 8
 - cambiare nome, 207-208
 - cluster, informazioni tramite le viste, 631
 - controllo delle definizioni, 108-109
 - definizioni, 332
 - di caratteri, larghezza, 332-333
 - di tipo data, 163
 - di unione, 814
 - flag, 377
 - in if, then, else, 327-329
 - indicizzate, informazioni tramite le viste, 629-630
 - informazioni sui commenti, 626-627
 - informazioni tramite le viste, 616-618
 - intestazioni, 101
 - locatori, 511
 - modifica, 341
 - modifica di intestazione e formato, 94
 - nomi delle, 332
 - NOT NULL, 52-53
 - NULL, 52-53
 - numeriche, 142, 276
 - policy, 576
 - riempimento, 118
 - selezione, 320
 - significato, 7
 - unioni, 213
 - valore NULL, 333
 - varietà negli indici, 376-377
 - vincoli di controllo, 338
 - vincolo, 814
 - informazioni tramite le viste, 624-625
 - virtuale, 1026
 COLS, sinonimo, 616

- COLUMN, comando, 88, 94, 108-109, 264-267, 814
Comandamenti della progettazione, 692
Comandi, 816
 annidati, 398
 creazione, 397
 di sistema privilegiati, 816
Combinazioni di tabelle, 62-63
COMFORT, tabella, 152
COMMENT, 817
Commenti, 760
 in SQLPLUS, 90
COMMIT (forma 1, interno SQL), 817
COMMIT (forma 2, istruzione PL/SQL), 817
Commit impliciti, 289
COMPARE, funzione di DBMS_LOB, 515, 523-524
Compilazioni, 467-468
COMPILE, clausola, 468
Compiti
 comprensione, 662-666
 profilo, 664-666
COMPLEANNO, tabella
COMPLETE, opzione del comando create snapshot, 536
COMPUTE, comando, 818
 calcoli possibili, 268
COMPUTE SUM, comando, 88, 97
 coordinazione con break on, 259-260
Computer remoto, 820
Comunicazione, protocollo, 817
CONCAT, comando, 820
CONCAT, funzione, 115
Concatenazione, 117, 820
Condizione, 820
 d'errore, personalizzazione, 445-447, 461-462
Configurazioni multimaster, 534
Confronti, funzione COMPARE, 524
CONNECT (forma 1), 820
CONNECT (forma 2, interno SQL), 820
CONNECT, comando, 359
CONNECT, ruolo, 355, 408
CONNECT BY, clausola, 246-251
 nelle query, 739-740
 regole di utilizzo, 254-255
CONNECT BY, operatore, 821
Connessione, 822
 a database remoti, 415-416
 tipi, 409
CONSTRAINT, clausola, 336, 338
CONTAIN, procedura del package CTXQUERY, 585
CONTAINS
 funzione, 558
 operatore, 822
Contemporaneità, 822
ConText, 555, 822
 caricamento dei dati, 583-584
 chiavi di testo, 575
 funzioni di espansione etimologica, 564
 impostazione, 571
 impostazioni delle tabelle, 575-582
 operatori di ricerca, 568
 punteggi, 558
 ricerche di testi, 558-568
 server, 822
 servizi linguistici, 586-588
Conversione, funzioni, 822
Conversioni automatiche tra tipi di dati, 192-195
CONVERT, funzione, 189, 824
Copia e incolla, 304-308
COPY
 comando, 414-415, 824
 procedura di DBMS_LOB, 515, 529
COPYCOMMIT, 414, 825
COS, funzione, 140, 150, 825
COSH, funzione, 140, 150, 826
Costi di lavorazione, 20
Costruttori, 81
COUNT, funzione, 140, 151, 156, 826
CREATE, opzione del comando copy, 415
CREATE ANY INDEX, privilegio, 533
CREATE ANY PROCEDURE, privilegio, 452, 458
CREATE ANY SNAPSHOT, privilegio, 532-533
CREATE ANY TABLE, privilegio, 533, 549
CREATE ANY TRIGGER, privilegio, 437, 549
CREATE ANY VIEW, privilegio, 533
CREATE CLUSTER, comando, 385, 826
CREATE CONTROLFILE, comando, 828
CREATE DATABASE, comando, 828
CREATE DATABASE LINK
 comando, 830
 privilegio, 364, 408, 533
CREATE DIRECTORY, comando, 512, 831
CREATE FUNCTION, comando, 114, 456-458, 831

- CREATE INDEX**
 comando, 349, 373-374, 532, 832
 privilegio, 532
- CREATE LIBRARY**, comando, 456, 833
- CREATE OR REPLACE FUNCTION**, comando, 348
- CREATE OR REPLACE PROCEDURE**, comando, 455
- CREATE OR REPLACE TRIGGER**, comando, 449
- CREATE OR REPLACE TYPE**, comando, 472
- CREATE OR REPLACE VIEW**, comando, 65, 343
- CREATE PACKAGE**, comando, 463-465, 833
- CREATE PACKAGE BODY**, comando, 463-465, 834
- CREATE PROCEDURE**
 comando, 455-456, 834
 privilegio, 452, 458
- CREATE PROFILE**, comando, 354, 835
- CREATE PUBLIC DATABASE LINK**, privilegio, 408
- CREATE ROLE**
 comando, 364, 836
 privilegio, 364
- CREATE ROLLBACK SEGMENT**, comando, 836
- CREATE SCHEMA**, comando, 837
- CREATE SEQUENCE**, comando, 386-387, 837
- CREATE SESSION**, privilegio, 364, 408
- CREATE SNAPSHOT LOG**, comando, 548-549, 839
- CREATE SNAPSHOT**
 comando, 535-538, 838
 privilegio, 532
- CREATE SYNONYM**
 comando, 361-362, 840
 e link, 406
 privilegio, 364
- CREATE TABLE**, comando, 332, 840
 clausola organization index, 347
 vincoli, 336
- CREATE TABLE**, privilegio, 353, 532, 549
- CREATE TABLE AS**, comando, 414
- CREATE TABLE AS SELECT**, comando, negli snapshot, 534
- CREATE TABLESPACE**, comando, 382-383, 844
- CREATE TRIGGER**
- comando, 440-441, 844
 privilegio, 437, 549
- CREATE TYPE**, comando, 77-78, 471, 845
- CREATE TYPE BODY**, comando, 483, 847
- CREATE UNIQUE INDEX**, comando, 375
- CREATE USER**, comando, 354, 847
- CREATE VIEW**, comando, 64, 848
 e link, 406-407
- CREATE VIEW**, privilegio, 364, 532
- CREATE_INDEX**, procedura del package CTX_DDL, 579
- CREATE_POLICY**, procedura del package CTX_DDL, 576
- CRT**, file, 850
- CTS_ALL_SERVERS**, vista, 573
- CTX_LING**, package, 587
- CTX_PREFERENCE_USAGE**, vista, 577
- CTX_SETTINGS**, tabella, 574
- CTX_TEMP**, tabella, 585
- CTX_USER_POLICIES**, vista, 580
- CTXADMIN**, ruolo, 575
- CTXAPP**, ruolo, 575
- CTXCTL**, utilità, 572
- CTXCTL**, utilità, 850
- CTXLOAD**, utilità, 584
- CTXQUERY**, package, 585
- ctxset.dmp**, file di ORACLE, 574
- CTXSRV**, utilità, 580
- CTXSYS**, utente, 575
- CTXUSER**, ruolo, 575
- CURRVAL**, 850
- CURSOR**, PL/SQL, 851
- Cursore**, 850
 apertura, 423
 attività, 754
 attributi, 427-428
 chiusura, 424
 coordinate, 851
 definizione, 422
 descrittore, 851
 ID, 851
 nei cicli, 431-432
 SQL, 851

D

- D**, formato delle date, 177
- Database Administrator (DBA)**, 853
- Database**, 852
 accessi pubblici, 370
 aggiornamenti remoti, 407-408

- Database (*continua*)
amministratore, 356
aperto, 853
apertura, 921
cancellazione degli utenti, 357
chiuso, 853
concessione di risorse, 370
connessioni esplicite, 409
connessioni predefinite, 409
copia di dati 413-415
creazione, 850
creazione di ruoli, 363-364
distribuito, 853
esempi relazionali, 13-16
file, 852
frammentato, 853
implementazione di oggetti, 605
impostazione per ConText, 571-575
link, 403-410, 852
locale, 853
montaggio, 921
nome, 852
oggetto, 852
ORDBMS, 69
orientato agli oggetti, 69
posizionamento degli indici, 379-380
RDBMS, 6
relazionale, 11, 69
 rischi, 17-18
relazionale a oggetti, 5, 69
remoto, 853
 connessione, 415
sistema, 853
smontato, 851
specifica, 853
struttura, 380-384
tabelle, 5
 tablespace, 380-384
Datafile, 381
Datazione di fatture, 316-318
Datazione Giuliana, 854
Date, 854
 aritmetica delle, 163-165
 e stringhe di caratteri, 398
 elenco formati, 177
 formati, 854
 formattazione, 173-177
 funzioni, 166, 856
 fusi orari, 166
 giorno della settimana, 168
 inserimento di ore, 286
 nelle clausole where, 182-183
ultimo giorno del mese, 168-169
DATE, tipo di dati, 114, 163, 187
Dati, 6-7
 aggiornamento nelle tabelle, 285-293
 astratti, 471-476
 binari RAW e formattati, 527
 blocco, 514
 cambio dell'ordinamento, 297
 caricamento tramite ConText, 583-584
 categorie, 23
 cattura, 662
 classi, 75
 collocazione, 505
 commit, 289
 comprensione, 667-669
 conservazione dei duplicati, 444-445
 conversioni automatiche tra tipi, 192-195
 descrizione dei tipi, 114
 di oggetti, 71
 disfunzioni, 24
 dizionario, 79
 elaborazione, 6, 662
 mediante viste oggetto, 479
 ereditarietà, 423
 errori di inserimento, 23
 file, 857
 gestione dei tipi LOB, 509-530
 gestione di grandi quantità, 504-505
 indipendenza, 857
 inserimento, 6, 285-288, 670
 maiuscole e minuscole, 35
 memorizzazione, 6
 nei cluster, 387
 modello logico, 25
 modifica, 285-293
 navigazione, 29
 normalizzazione, 16, 18, 24-28
 protezione nelle viste, 344
 recupero, 6
 tramite i file di redo log, 345
 relazioni logiche, 48
 relazioni normali, 18
 remoti
 accesso, 403-418
 aggiornamento, 407-408
 copie, 531
 restituzione, 662
 rollback, 288-290
 selezione con SQL, 42
 selezione da array variabili, 494-495
 selezione da tabelle oggetto, 592
 sincronizzazione con i trigger, 444

- spostamento da un sistema vecchio al nuovo, 308
 strutture incapsulate, 76
 tipi, 858
 - astratti, 72-73
 - standard, 72
 tipo BFILE, 507-508
 tipo BLOB, 507-508
 tipo CLOB, 507-508
 tipo LOB, 507-508
 tipo LOB, 74
 tipo LONG RAW, 507
 tipo LONG, 507
 tipo NCLOB, 507
 trasparenza delle locazioni, 411
 visualizzazione tramite snapshot, 549-551
- DAY, formato delle date, 177
 DBA (Database Administrator), 853
 - privilegi, 859
 DBA, ruolo, 356-357, 408
 DBA_CATALOG, vista, 612
 DBA_CLU_COLUMNS, vista, 631
 DBA_COLL_COMMENTS, vista, 627
 DBA_CONS_COLUMNS, vista, 624
 DBA_CONSTRAINTS, vista, 623
 DBA_DIRECTORY, vista, 635
 DBA_ERRORS, vista, 641
 DBA_IND_COLUMNS, vista, 630
 DBA_INDEXES, vista, 629
 DBA_OBJECT_SIZE, vista, 642
 DBA_SEQUENCE, vista, 621
 DBA_SNAPSHOTS, vista, 638
 DBA_SOURCE, vista, 467
 DBA_SOURCE, vista, 641
 DBA_SYNONYMS, vista, 621
 DBA_TAB_COLUMNS, vista, 617
 DBA_TAB_COMMENTS, vista, 626
 DBA_TABLES, vista, 615
 DBA_TRIGGERS, vista, 640
 DBA_VIEWS, vista, 620
 DBMS_LOB, package, 515-529
 DBMS_OUTPUT, package, 459-460
 DBMS_REFRESH, package, 544-547
 DBMS_SNAPSHOT, package, 543
 DBS_DB_LINKS, vista, 636
 DBWR, processo, 859
 DCL (Data Control Language), 852
 DD, formato delle date, 177
 DDD, formato delle date, 177
 DDL (Data Definition Language), 852
 DDL, server, 574
 Deadlock, 859
 Debugging, generazione di informazioni, 460
DECLARE
 - comando, 860
 - parola chiave nei blocchi PL/SQL**DECLARE CURSOR**, comando (forma 1, interno SQL), 860
DECLARE CURSOR, comando (forma 2, PL/SQL), 860
DECLARE DATABASE, comando, 861
DECLARE STATEMENT, comando, 861
DECLARE TABLE, comando, 861
DECODE, funzione, 189, 197, 305, 315-330
 - maggiori, minori e uguali, 329-330
 - utilizzo di MOD, 323-325**Default**, 119, 862
DEFAULT ROLE, clausola del comando alter user, 366
DEFINE, comando (SQL*PLUS), 275, 863
DEFINE_EDITOR, comando, 88, 106-107
DEL, comando, 863
DELETE (forma 1, PL/SQL), 863
DELETE (forma 2, comando SQL), 864
DELETE (forma 3, interno SQL), 865
DELETE, comando, 103-104, 291-292
 - problemi, 295**DELETE**, privilegio, 353
 Denormalizzazione, 674-682
DREF, operatore, 594-596, 602, 865
DESC, parola chiave, 47
DESCRIBE (forma 1, comando SQL*PLUS), 865
DESCRIBE (forma 2, interno SQL), 43, 52, 79, 411, 617, 865
 Descrittori di connessione SQL*Net, 635
DESTROY, procedura del package DBMS_REFRESH, 546-547
 Deviazione standard, 155
Device, 866
 Diagrammi a barre, 298-300
 - orizzontale, 299
 Dichiarazioni, 866
 - dei blocchi PL/SQL, 420-421**DICT**, sinonimo, 611
DICT_COLUMNS, vista, 611
DICTIONARY, vista, 610-611
Directory, 866
 - creazione, 512**DISABLE**, clausola, 866
DISABLE ALL TRIGGERS, clausola del comando alter trigger, 448
DISCONNECT, comando, 867

- DISTINCT, clausola, 156-157
Divisione, 143, 761
Dizionario di dati, 79, 609, 867
 blocchi, 867
 blocchi della cache, 867
 blocchi operativi, 867
 cache, 868
 nomenclatura, 610
DML (Data Manipulation Language), 852, 868
DML, server, 574
DNS, struttura, 417
DOCUMENT, comando, 868
Domain Name Service, struttura, 417
Domini di rete, 417
Doppioni, eliminazione, 136
DROP ANY PROCEDURE, privilegio, 469
DROP ANY SNAPSHOT, privilegio, 552
DROP ANY TABLE, privilegio, 553
DROP ANY TRIGGER, privilegio, 449, 553
DROP CLUSTER, comando, 869
DROP DATABASE LINK, comando, 869
DROP DIRECTORY, comando, 869
DROP FUNCTION, comando, 469, 870
DROP INDEX, comando, 870
DROP LIBRARY, comando, 870
DROP PACKAGE, comando, 469, 870
DROP PROCEDURE, comando, 469, 870
DROP PROFILE, comando, 871
DROP ROLE, comando, 367-368, 871
DROP ROLLBACK SEGMENT, comando, 871
DROP SEQUENCE, comando, 871
DROP SNAPSHOT
 comando, 872
 privilegio, 552
DROP SNAPSHOT LOG, comando, 872
DROP SYNONYM, comando, 872
DROP TABLE
 comando, 339, 872
 privilegio, 553
DROP TABLESPACE, comando, 873
DROP TRIGGER
 comando, 449, 873
 privilegio, 553
DROP TYPE, comando, 873
DROP TYPE BODY, comando, 873
DROP USER, comando, 357, 874
DROP VIEW, comando, 874
DROP_INDEX, procedura del package CTX_DDL, 578, 583
DUAL, tabella, 165, 874
DUMP, comando, 874
DY, formato delle date, 177
- E**
- EBCDIC, 875
Eccezioni, 446
 gestione nei blocchi PL/SQL, 434-436
 informazioni tramite le viste, 625
ECHO
 comando (SQL*PLUS), 875
 parola chiave, 274
EDIT, comando, 88, 875
EDITOR, variabile in SQLPLUS, 272
Editor, 105
Editor della riga di comando, 90, 101-102
 cancellazione di righe, 103
 riga corrente, 104
Elaborazione distribuita, 875
Elenchi di valori, 140
 restituzione, 60
 test logici, 55
 unione, 228-233
EMBEDDED, comando (SQL*PLUS), 876
EMPTY_BLOB, funzione, 512-514
EMPTY_CLOB, funzione, 512-513
ENABLE, opzione, 876
ENABLE ALL TRIGGERS, clausola del comando alter trigger, 448
END
 istruzione, 876
 clausola del comando create type body, 485
 nei blocchi PL/SQL, 434
END LOOP, parola chiave, 426
Enqueue, 876
Entità, 8, 877
Entità-relazioni, modello, 877
Equi-join, 877
ERASE, procedura di DBMS_LOB, 515, 527-528
Ereditarietà
 delle tabelle, 345
 di implementazione, 76
Errori, ricerca nelle procedure, 459-460
Esadecimale, notazione, 877
Esadecimali, valori, 325
ESCAPE (SQL*PLUS), 877
Esecuzione, 884
 ordine di, 204-205
Esponenziale, 759

- Espressione, 877
 logica, 877
- Estensioni
 libere, 877
 usate, 877
- EXCEPTION**
 istruzione, 877
 parola chiave, 434, 462-463
- EXCEPTION_INIT**, funzione, 880
- EXCEPTIONS**, tabella, 625
- EXECUTE**
 comando (dinamico interno SQL),
 452-453, 881
 privilegio, 452
- EXECUTE ANY PROCEDURE**, privilegio,
 454
- EXECUTE IMMEDIATE**, comando
 (dinamico interno SQL), 882
- EXISTS**, operatore, 882
 in sottoquery correlate, 222-223
- EXIT**
 comando (forma 2, SQL*PLUS), 883
 funzione (forma 1, PL/SQL), 883
- EXIT WHEN**, comando, 426
- EXP**, comando, 140, 147, 884
- EXP_FULL_DATABASE**, ruolo, 356
- EXPLAIN PLAN**, comando, 737, 884
- EXPORT**, utilità, 884
- EXTERNAL PROGRAM**, clausola del
 comando create procedure, 456
- EXTERNALLY**, opzione del comando
 create user, 354
- F**
- Fase di definizione, 863
- Fase di descrizione, 866
- FAST**, opzione del comando create snapshot,
 536
- Fattore umano, 661-672
- FATTURA**, tabella, 316
- FEEDBACK** (SQL*PLUS), 885
- FETCH**, 885
- FETCH**, comando (forma 1, interno SQL),
 424, 885
- FETCH**, istruzione (forma 2, PL/SQL),
 885
- File, 384, 886
 di controllo, 887
 init.ora, 574-575
 numero massimo di aperti, 516
 tipi, 886
- File di avvio, 89
 311-313
 creazione, 397-400
- File di comandi, esecuzione, 108
- File di dati, 857
- File di redo log, 345
- FILECLOSE**, procedura di DBMS_LOB,
 515, 530
- FILECLOSEALL**, procedura di
 DBMS_LOB, 515, 530
- FILEEXISTS**, procedura o funzione di
 DBMS_LOB, 515, 530
- FILEGETNAME**, procedura di
 DBMS_LOB, 516, 530
- FILEISOPEN**, procedura di DBMS_LOB,
 516, 530
- FILEOPEN**, procedura di DBMS_LOB, 515,
 530
- Filtri di ConText, 578
- FLOOR**
 comando, 146, 887
 valore, 140
- FLUSH** (SQL*PLUS), 887
- fm, suffisso per le date, 178
- Foglie di alberi, 246, 887
- FOLD_AFTER**, comando, 282-283
- FOLD_BEFORE**, comando, 282-283
- FOR**, cicli, 429-431, 887
- FOR EACH ROW**, clausola del comando
 create trigger, 441
- FOR UPDATE**, clausola, 514
- FORCE**, opzione del comando create
 snapshot, 536
- Formattazioni
 avanzate, 257-260
 di numeri, 95, 275-277
 elenco, 277
- FormFeed**, 887
- FROM**
 clausola, 45, 887
 parola chiave, 9
- Front-end, 415-416
- FULL TABLE SCAN**, metodo, 888
- Funzioni, 451, 888
 a valori singoli, 142
 combinazioni con funzioni di gruppo,
 152
 aggiunta, 123-124
 annidate, 154
 combinazioni, 121-123
 compilazione, 467-468
 di conversione, 187-195, 822

- F**
- Funzioni (*continua*)
 - di data, 166
 - combinazioni, 171-173
 - di elenco, 157-158, 888
 - di espansione etimologica, 564
 - di gruppo, 150-152, 888
 - clausola order by, 212-213
 - di hash, 631
 - di ordinamento, 297
 - di trasformazione, 195-198
 - e procedure, 454
 - informazioni tramite le viste, 640-642
 - ipertestuali, 895
 - nomi, 805
 - numeriche, 927
 - classi, 139-141
 - opzioni, 115
 - personalizzate, 460-461
 - scaricamento, 469
 - sostituzioni, 468
 - statistiche, 151
 - stringa, 113-115
 - clausola order by, 134-135
 - clausola SOUNDEX, 136
 - clausola where with, 134-135
 - suddivisione per tipi di dati, 188-189
 - trigonometriche, 150
 - utilizzo, 805
 - Fusi orari, 166, 179
 - Fuzzy, 889
 - coincidenze nelle ricerche tramite ConText, 565-566
- G**
- Generatori di codici, 87
 - Genitore, 889
 - Gestore di blocchi a livello di riga, 889
 - GET, comando, 279-280, 889
 - GETLENGTH, funzione di DBMS_LOB, 515, 522-523
 - GIORNALE, tabella, 42
 - Gist, 586, 889
 - GLB, funzione, 890
 - GLOBAL, clausola del comando create index, 349-350
 - GLOGIN.SQL, file, 107
 - GOTO, istruzione, 433-434, 890
 - Grafici, 298-300
 - GRANT, comando (forma 1, ruoli e privilegi di sistema), 890
- H**
- GRANT, comando (forma 2, privilegi degli oggetti), 891
 - per la concessione di privilegi, 364
 - GRANT ANY PRIVILEGE
 - privilegio, 353
 - ruolo, 364
 - GREATEST, funzione, 140, 158, 306, 892
 - nell'aritmetica delle date, 166-167
 - GREATEST_LB, funzione, 892
 - GROUP BY, clausola, 201-203
 - 892
 - invece di order by, 343
 - Gruppi di aggiornamento di snapshot, 544-547, 893
 - Gruppi di valori, 139-140
- I**
- Hardware proprietario, 4
 - Hash, cluster, 386
 - Hash, funzioni, 631
 - HASH IS, clausola del comando create cluster, 631
 - HASH JOIN, 726-731
 - HAVING, clausola, 201-203, 893
 - logica, 211
 - HEADING (SQL*PLUS), 893
 - HEADSEP (SQL*PLUS), 894
 - default, 93
 - HELP, comando, 894
 - HEXTORAW, funzione, 189, 195, 894
 - HH, formato delle date, 178
 - HH12, formato delle date, 178
 - HH24, formato delle date, 178
 - Highwatermark, 288
 - Hint, 288, 695-696, 704-705, 742, 894
 - e parametri, 719
 - HOCKEY, tabella, 372
 - Host, 403
 - processi, 400-401
 - HOST, comando, 88, 106-107, 397
 - HTF, libreria, 895
 - HTP, libreria, 900
- L**
- I, formato delle date, 177
 - ID utente, 40
 - IDENTIFIED BY
 - clausola del comando set role, 367
 - del comando alter role, 365-366
 - IDENTIFIED EXTERNALLY, clausola del comando alter role, 366

- IF, istruzione, 906
 IF, THEN, ELSE, logica, 315-316, 424-425
 IMP_FULL_DATABASE, ruolo, 356
 IMPLICIT DESTROY, parametro della procedura MAKE, 547
 IMPORT, utilità, 906
 IMS, 22
 IN
 operatore, 55, 61, 906
 IN OUT, qualificatore, 455
 IND, sinonimo, 627
 INDEX RANGE SCAN, 697
 INDEX UNIQUE SCAN, 697
 Indicatore, variabile, 907
 Indice, 371-380, 696, 706, 908
 bitmap, 375-376
 compresso, 908
 concatenato (o chiave), 908
 creazione, 349, 373-374
 creazione automatica per gli snapshot, 536
 delle tabelle di snapshot, 540-541
 formato B-tree, 374
 globale, 349-350, 909
 informazioni tramite le viste, 627-630
 locale, 909
 messa a punto, 705-706
 posizionamento nel database, 379-380
 quantità, 378
 ricostruzione, 380
 segmento, 909
 unici, 345, 375, 909
 varietà di colonne, 376-377
 Indici cluster, 378
 Indici testuali, 558
 creazione, 579-582
 ottimizzazione, 583
 scaricamento, 582-583
 INDIRIZZO, tabella, 127
 INDIRIZZO_TY, tipo, 80
 Informazioni
 estrapolazione, 29
 organizzazione, 31
 sicurezza, 353
 spooling, 99
 tabelle, 8
 init.ora, file, 909
 modifiche, 574-575
 INITCAP, funzione, 115, 124-125, 909
 INPUT, comando, 104, 279, 909
 Inserimenti
 arrotondamento, 334-336
 con sottoquery, 513
 Inserimento
 di dati, 6
 di informazioni, 279
 di record, 81
 INSERT
 comando, 910
 opzione del comando copy, 415
 INSERT AS SELECT, comando, 501
 con i valori LOB, 513
 INSERT INTO...SELECT FROM, sintassi, e i tipi LONG, 287
 INSTEAD OF, trigger, 345, 439-440, 480-486, 911
 INSTR, funzione, 115, 129-133, 911
 di DBMS_LOB, 515
 INSTRB, comando, 912
 Integrità
 a livello dei nomi, 684-686
 referenziale, 911, 957
 Interruzioni:, 96
 INTERSECT, comando, 710-714, 912
 INTERVAL, parametro delle procedure di DBMS_REFRESH, 546
 Intestazione
 di colonna, 101
 di riga, 912
 IS, operatore, 53-54
 IS NOT NULL, operatore, 51
 IS NULL, operatore, 51, 53, 912
 ISO, 178
 Istanza, 913
 identificativo, 914
 Istogramma, 914
 sui valori delle colonne, 617-618
 Istruzione, 914
 eseguibile SQL, 914
 IW, formato delle date, 178
 IY, formato delle date, 177
 IYY, formato delle date, 177
 IYYY, formato delle date, 177
- J**
- J, formato delle date, 178
 JOB_QUEUE_PROCESSES, parametro di init.ora, 542
 Join, 913
- K**
- K, suffisso, 383
 Kernel, 914

L

Larghezza delle colonne, 333
LAST_DAY, funzione, 166, 168-169, 914
LAVORATORE, tabella, 26
LAX, parametro delle procedure di DBMS_REFRESH, 545
LEAST, funzione, 140, 158, 914
nell'aritmetica delle date 166-167
LEAST_UB, funzione, 915
LENGTH, funzione, 115, 125-126, 307, 915
LENGTHB, funzione, 915
Lettura ripetibile, 915
LEVEL, pseudocolonna, 247, 915
LGWR (LOG WRITER PROCESS), 918
Librerie, 916
creazione, 456
LIKE, operatore, 50-52, 129, 916
Limitazione, 916
LINESIZE, (SQL*PLUS), 93, 97, 916
Linguaggi naturali, 19
Lingue nazionali, supporto, 137-138
Linguistic, server, 574
Link di database, 403-410
aggiornamento dati remoti, 407-408
e sinonimi, 406-407
e viste, 406-407
gestione dei nomi, 416-417
informazioni tramite le viste, 635-636
privati, 404
sintassi, 408
pubblici e privati, 408
pubblici, 404
LIST, comando, 102, 104
LN, funzione, 140, 147, 917
LOB (Large Object), tipo di dati, 74, 507, 917
aggiornamento, 514
chunk, 510
definizione dei parametri di memorizzazione, 509-511
eliminazione, 530
gestione e selezione, 511-513
informazioni tramite le viste, 634-635
logging, 511
parametri di memorizzazione, 509-511
pctversion, 510
LOB, clausola del comando create table, 509
LOCAL, parola chiave del comando create index, 349
Locatori, 511-512
LOCAZIONE, tabella, 60
LOCK TABLE, comando, 917

Lock, 514
LOG, funzione, 140, 147, 918
Log di snapshot, 536, 539, 548
informazioni tramite le viste, 638
modifica, 551-552
scaricamento, 552-553
LOG WRITER PROCESS (LGWR), 918
Logaritmi, 147
LOGGING, parametro, 511
Logica condizionale, nei blocchi PL/SQL, 424-426
Login account, 918
LOGIN.SQL, file, 107
Logon account, 918
LONG, tipo di dati, 114, 507, 918
limitazioni, 345
LONG RAW, tipo di dati, 114, 507, 918
LOOP, parola chiave, 426
LOWER, funzione, 115-116, 124-125, 918
LPAD, funzione, 115, 118-119, 298
LTRIM, funzione, 120, 919
LUB, funzione, 919

M

M, suffisso, 383
MAKE, procedura del package DBMS_REFRESH, 544-545
MAKE_REF
funzione, 919
operatore, 601-602
Marcatori di posizione, 51
Mask.sql, file, 278
Master, tabelle, 534
MATEMATICA, tabella, 142
Matrici, calcolo, 322
MAX, funzione, 140, 158-159, 919
MAXDATA (SQL*PLUS), 920
MAXVALUE, parola chiave, 349
MAXEXTENTS, parola chiave del comando create tablespace, 384
Meccanismi di blocco dei file, 920
Medie ponderate, calcolo, 269
Memorizzazione
delle tabelle, 385
di dati, 6
esterne, 576
fuori linea, 508-509
interne, 575
interne master-dettaglio, 575-576
MERGE JOIN, 721-723

- Metodi, 920
 collettore, 492
 costruttori, 472, 474
 creazione, 483-485
 dei tipi di dati astratti, 474
 degli oggetti, 71, 75
 gestione, 486-487
 mappa, 486
 ordine, 486
- MI, formato delle date, 178
- MIN, funzione, 140, 158-159, 920
- MINEXTENTX, parola chiave del comando create tablespace, 384
- MINUS, 710-714, 920
 operatore di ConText, 561, 568
- MIPS, 20
- MM, formato delle date, 177
- MOD, funzione, 140, 146, 323-325, 921
- Modalità esclusiva, 921
- Modello relazionale, 11
 rigido, 73
- MODIFY, clausola, 340
- Moltiplicazione, 143, 759
- MON, formato delle date, 177
- MONTH, formato delle date, 177
- MONTHS_BETWEEN, funzione, 166, 171, 921
- Multimaster, configurazioni, 534
- Multiprocesso, 921
- MULTISET, parola chiave, 502
- N**
- National Language Support, 137-138
- NCLOB, tipo di dati, 74, 507
- NEAR, operatore, 922
 nelle ricerche di ConText, 563, 568
- NESTED LOOPS, 723-726
- Net8, stringhe di connessione, 409-410
- NEW, parola chiave del comando create trigger, 441
- NEW_LINE, funzione di DBMS_OUTPUT, 460
- NEW_TIME, funzione, 166, 179-180, 922
- NEWLINE, comando, 396
- NEWPAGE (SQL*PLUS), 922
- NEXT, clausola del comando create snapshot, 542
- NEXT_DATE, parametro delle procedure di DBMS_REFRESH, 546
- NEXT_DAY, funzione, 166, 168, 922
- NEXTVAL, pseudocolonna, 922
- NLS_DATE_FORMAT, parametro, 173, 185
- NLS_INITCAP, funzione, 922
- NLS_LOWER, funzione, 923
- NLS_UPPER, funzione, 923
- NLS-DATE-LANGUAGE, parametro, 173
- NLSSORT, funzione, 922
- NO_DATA_FOUND, eccezione
- NOAUDIT, comando (forma 1, oggetti schema), 923
- NOAUDIT, istruzione (forma 2, istruzioni SQL), 924
- NOCACHE, parametro, 510
- Nodi di alberi, 246
- Nodo, 924
 terminale, 924
- NOLOGGING, opzione, 345, 511, 924
- NOME, tabella, 304
- Nomi
 degli oggetti, 925
 delle funzioni, 805
 e loro utilizzo, 822
 degli snapshot, 539-540
 degli utenti, 353
 dei trigger, 447
 dei vincoli, 338
 delle colonne, 332
 delle tabelle, 332
 di procedure, 463
 di servizio, 409-410, 416-417
 lingua corrente, 34
 maiuscole e minuscole, 35
 normalizzazione, 35-36
 problemi di assegnamento, 33
- Non-equi-join, 925
- Normalizzazione, 16, 18, 24-28, 31, 673
 dei nomi degli oggetti, 683-690
 forme, 25-28
- NOT BETWEEN, operatore, 55
- NOT BETWEEN... AND, operatore, 55
- NOT EXISTS, operatore, 224, 925
- NOT IN, 54, 55
 sostituzione con NOT EXISTS, 227-228
 sostituzione con unioni esterne, 225-226
- NOT
 clausola, 53
 operatore, 925
- NOT NULL
 parametro, 43
 valore, 52-53
- NULL
 istruzione (forma 1, PL/SQL), 926
 parametro, 43

- NULL (continua)
 valore di colonna, 926
 nelle date, 286
 nelle funzioni di gruppo, 150
 nelle funzioni numeriche, 143-144
- NUMBER, tipo di dati, 48, 114, 187
 precisione, 332-335
- Numeri, 114
 ampiezza, 45
 arrotondamento, 148-149
 calcolo del logaritmo, 147
 calcolo del modulo, 146
 calcolo della media, 150-151
 elenco formattazioni, 277
 elevamento a potenza, 146
 esponenziale, 147
 formati, 928
 formattazione, 95, 275-277
 funzioni, 927
 numeriche, 139-141
 trigonometriche, 150
 precisione, 333-335
 problemi di formattazione, 259
 radice quadrata, 147
 segno, 149
 tipo di dati, 928
 valori assoluti, 145
- Numero di versione, 929
- Numero sequenziale di riga, 929
- NUMWIDTH (SQL*PLUS), 45, 929
- NVL, funzione, 140, 144-145, 929
- O**
- OBJ, sinonimo, 613
- OCI, 511
- Oggetti, 930
 annidati, 589
 caratteristiche, 75-76
 colonna, 75, 589-590
 convenzioni di nomenclatura, 76-77
 dati, 71
 ereditarietà, 76
 esempio comune, 77-78
 implementazione nel database, 605
 incorporati, 75
 informazioni tramite le viste, 613
- LOB, 506
- locali e remoti, 539-540, 549
- nomi, 925
- obbligo di utilizzo, 69
- privati, 464
- referenziati, 75, 589
- riga, 75, 589-590
- strutture orientate agli, 476
- strutture semplici, 78-79
- svantaggi, 71
- utilità, 70
- OID (Object Identifier), 590, 592, 930
 generazione, 600-601
- OLD, parola chiave del comando create trigger, 442
- ON DELETE CASCADE, clausola, 337
- OO (Object Oriented), 69, 476
- OPEN, comando, 423, 930
- OPEN CURSOR, comando (interno SQL), 931
- OPEN-LINKS, parametro di init.ora, 405
- Operatori, 931
 di query, 932
 di ricerca testuale, 932
 di soglia, 559
 insiemistici, 933
 logici, 933
 relazionali, 932
 sintattici, 934
 logici, 47
 a valori multipli, 60
 a valori singoli, 59
 combinazione, 56
 precedenza, 160
 precedenza, 57
- OPS\$ LOGIN, nome utente, 932
- OPTIMIZE_INDEX, procedura del package CTX_DDL, 583
- Opzioni, 115
- OR, operatore booleano, 54, 935
 combinazione con AND, 56-57
 nelle ricerche tramite ConText, 561, 568
- ORACLE PROGRAM INTERFACE (OPI), 935
- ORACLE WEB APPLICATION SERVER, 936
- ORACLE WEB SERVER, 936
- Oracle*Names, 416-417
- ORACLE
 applicazioni, 19
 autorità, 353-370
 ConText, 555-588
 creazione di tavole, 331-332
 dati, 6-7
 dizionario dei dati, 609-658
 e lingue nazionali, 137-138
 elaborazione dei dati, 6

- funzione DECODE, 315-330
 funzione degli indici, 34-346
 indici, 371-380
 metodi costruttori, 472
 modello relazionale, 4
 modifica dei dati, 285-293
 modifica dell'ambiente, 371-388
 opzione distribuita, 531
 ottimizzatore, 693-743
 package DBMS_SNAPSHOT, 543
 ruoli, 353
 - standard, 355-356
 schema dei file, 381
 sicurezza, 353, 361
 tabelle, 5
 tipi di database, 69
- ORDBMS**, 7, 69, 936
- ORDER BY**, clausola, 47, 203-204, 936
 - con funzioni di gruppo, 212-213
 - con funzioni stringa, 134-135
 - con RowNum, 325-326
 - invece di group by, 343
 - nelle viste, 343
 - nelle funzioni, 297
- Ordinamento**, 10
 - cambio di, 297
 - descendente, 47
- ORGANIZATION INDEX**, clausola del comando create table, 347
- Orientamento agli oggetti**
 - analisi e progettazione, 84-85
 - astrazione, 75
 - polimorfismo, 76
 - vantaggi e svantaggi, 70-71
- OS_ROLE**, parametro di init.ora, 366
- Ottimizzatore**, 693-747, 937
 - CHOOSE**, 693
 - COST**, 693
 - RULE**, 693
- OUT**, qualificatore, 455
- OV**, suffisso, 476
- P**
- P.M.**, formato delle date, 178
- Package**, 451, 937
 - compilazione, 467-468
 - corpo, 464
 - creazione, 463-465
 - DBMS_OUTPUT, 459-460
 - e procedure, 454-455
- informazioni tramite le viste, 640-642
 inizializzazione, 465
 scaricamento, 469
 sostituzioni, 468
 specifica, 464
- PAGESIZE (SQL*PLUS)**, 98, 937
- Pagina**, 937
 - numero massimo di righe, 98
 - righe vuote, 98
- Parametri**, 937
 - variabili, 938
- Parentesi**, 938
- Parole chiave**, 9
 - e simboli PL/SQL, 938
- Parole riservate**, 941
- Parsing**, 943
- PARTITION BY RANGE**, clausola del comando create table, 348-349
- Partizione**, 347-351, 943
 - creazione, 348-349
 - creazione di indici globali, 349-350
 - gestione, 350-351
 - indicizzazione, 349-350
 - realizzazione di query, 350-351
- Password**, 40, 353, 943
 - aggiunta ai ruoli, 365-366
 - gestione, 354-355
 - rimozione dai ruoli, 366
- PASSWORD**, comando (SQL*PLUS), 355, 943
- PAUSE (forma 1, SQL*PLUS)**, 944
- PAUSE**, comando (forma 2, SQL*PLUS), 944
- PCTFREE**, 944
- PCTINCREASE**, parola chiave del comando create tablespace, 384
- PCTUSED**, 944
- PCTVERSION**, parametro, 510
- Percorso di esecuzione**, 732-737
- Percorso diretto**, 935
- PERSONA_TY**, tipo, 80
- Personalità**, 944
- pkREF**, riferimenti, 602
- PL/SQL**, 70
 - a oggetti, 603-605
 - etichette, 763
 - funzioni, 451
 - funzioni nei blocchi, 104
 - gestione delle eccezioni, 434-436
 - nozioni di base, 419-424
 - package, 451
 - procedure memorizzate, 451

- Policy, 576, 944
 visualizzazione delle preferenze, 577
- Polimorfismo, 76
- Pool SQL condiviso, 945
- POWER, funzione, 140, 146-147, 945
- Pragma, 945
- Precedenze, 57, 159-161, 945
- Precompilatore, 946
- Predicato, 946
- Preferenze delle policy, 577
- PREPARE, comando (interno SQL), 946
- Prestazioni
 e progettazione, 673-682
 miglioramento con APPEND, 287-288
 miglioramento tramite gli indici, 347
 miglioramento tramite snapshot, 531-532
- PRESTITO, tabella, 490
- Prima forma normale, 26-27
- PRIMARY KEY, vincolo, 337
- PRINT, comando, 947
- PRIOR, clausola, 947
- Privilegi, 947
 concessione a ruoli, 364
 concessione, 357
 revoca dai ruoli, 367
 revoca, 356-357
 trasmissione, 362-363
 violazione, 362
- PRO*C, 951
- PRO*COBOL, 951
- PRO*FORTRAN, 951
- Procedure, 951
 compilazione, 467-468
 creazione di sinonimi, 453
 e funzioni, 454
 e package, 454-455
 esecuzione, 452-454
 ipertestuali, 900
 informazioni tramite le viste, 640-642
 memorizzate, 451
 nomi, 463
 ricerca di errori, 459-460
 riferimenti a tabelle remote, 458-459
 scaricamento, 469
 sostituzioni, 468
- Processi
 host, 400-401
 numero, 542
 in background, 799
- PMON, 951
- server, 951
- singolo, 951
- PRODUCT_USER_PROFILE, tabella, 952
- Profili, 354-355, 952
- Progettazione
 dieci comandamenti, 683-692
 e prestazioni, 673-682
 orientata agli oggetti, 84-85
- PROMPT, comando, 393-394, 952
- PROPOSTA, tabella, 507, 508
 creazione, 509
- Protocolli di comunicazione, 404, 817
- Pseudocolonne, 66, 320, 412, 952
- Public, 953
 clausola di create database link, 408
 clausola del comando grant, 370
- Punteggi di ConText, 558
- PUT, funzione di DBMS_OUTPUT, 460
- PUT_LINE
 funzione di DBMS_OUTPUT, 460, 495
 procedura di DBMS_LOB, 517
- Q**
- Q, formato delle date, 178
- Quarta generazione, strumenti, 17-18
- Query ConText, 558
- Query, 9, 953
 a un solo passaggio, 584
 ad albero, 953
 combinazione, 58
 correlata, 953
 di intervallo, 628
 di testo, 557
 schema di elaborazione, 584
 distribuita, 953
 e cluster, 741-742
 e connect by, 739-740
 e link di database, 741
 e sequenze, 740-741
 generazione del codice, 389-395
 genitore, 953
 in due passaggi, 584
 principale, 953
 remote, 404-406
 semplice, 9-10, 533
 su tabelle annidate, 499-503
 sulle partizioni di tabelle, 350-351
 sulle viste oggetto, 322, 602
 unione di tabelle con where, 66-67
 velocizzazione con gli indici, 372
- QUIT, comando, 953
- Quota, 354, 954
 parametro di create or alter user, 370

R

- Radice, 962
 Raggruppamenti, esempio, 319-321
RAISE, istruzione, 954
RAISE_APPLICATION_ERROR, procedura, 445-447, 462-463
Rami, 954
RAW, tipo di dati, 114, 527, 954
RAWTOHEX, funzione, 189, 195, 954
RDBMS, 6, 955
READ, procedura di DBMS_LOB, 515-519
REBUILD, clausola del comando alter index, 380
Record, 955
 - inserimento, 81
 - inserimento in array variabili, 492-493
 - inserimento in tabelle annidate, 497-503**RECOVER**, comando, 955
RECSEP (SQL*PLUS), 956
RECSEPCHAR (SQL*PLUS), 956
Recupero
 - dei supporti, 956
 - di dati, 6
 - di istanza, 956**Redo log**, 345, 956
 - file, 345
 - fuori linea, 956
 - in linea, 956**REF**
 - indicatore, 75
 - operatore, 593-594
 - tipo di dati, 957**REFERENCES**, clausola, 337
REFRESH, procedura del package DBMS_REFRESH, 547
REFRESH, procedura di DBMS_SNAPSHOT, 543
REFRESH_ALL, procedura di DBMS_SNAPSHOT, 544
REFSNAP, utilità, 544
REGISTRO, tabella, 94, 384
Regola di integrità referenziale, 957
Relazionale, 11
 - sistema di gestione di database, 957
 - modello, 11
 - teoria, 8**Relazionale a oggetti**, sistema di gestione di database, 957
Relazione, 957
Relazioni logiche, 48
 - tra tabelle, 63**REMARK**, comando, 87, 90, 957
RENAME, comando, 957
REPFOOTER, comando, 93, 958
REPHEADER, comando, 93, 958
REPL, abbreviazione di replace, 105
REPLACE
 - funzione, 308-309, 959
 - opzione del comando copy, 415
 - opzione del comando save, 105**Replicazioni multimaster**, 537
Repliche di dati, 531
Report
 - aggiunta di viste, 262-263
 - creazione, 90-102, 257-277
 - dimensioni del corpo, 98
 - fasi dello sviluppo, 108
 - formattazione dei numeri, 275-277
 - formattazione dei titoli, 262
 - impiego di variabili, 273**REQUEST_GIST**, procedura del package CTX_LING, 587
REQUEST_THEMES, procedura del package CTX_LING, 587
RESOURCE, ruolo, 356, 437, 452
Rete, 959
RETURN, parola chiave del comando, create function, 457
RETURN NUMBER, clausola del comando create function, 457
REVOKE, comando (forma 1, privilegi di sistema e ruoli), 960
REVOKE, comando (forma 2, privilegi degli oggetti), 960
REVOKE ALL, comando, 368
Ricerche di testi
 - coincidenze fuzzy, 565-566
 - limitazione dei record restituiti, 569
 - parole con la stessa radice, 564-565
 - per esatta coincidenza di più parole, 560-562**Ricerche di testi (*continua*)**
 - per esatta coincidenza di una frase, 562-563
 - per esatta coincidenza di una parola, 559-560
 - per parole dal suono simile ad altre, 566-567
 - per prossimità, 563
 - utilizzo di caratteri jolly, 564**Ricorrenze**, 130
 - conteggio nelle stringhe, 308-309**Riferimenti**, 75

- Riga di comando, 816
Righe, 321
cancellazione, 353
cancellazione delle dipendenti, 337
conteggio, 323-324, 390
correnti, 825
filtro, 738-739
inserimento di vuote, 259-262, 282
inserimento in tabelle oggetto, 591-592
numero massimo di caratteri, 97
ordinamento, 706-707
raggruppamento, 708-709
rimozione, 291-292
saltare una o più, 268
significato, 7
suddivisione in partizioni, 347-351
tupla, 8
Risorsa, 960
Risorse del database, concessione, 370
RIVISTA, tabella, 131
RM, formato delle date, 177
RNDS, restrizione, 485
RNPS, restrizione, 485
Roll forward, 961
Rollback, 961
automatico, 289-290
differito, 866
segmenti, 414
ROLLBACK, comando (forma 1, SQL), 961
ROLLBACK, comando (forma 2, interno SQL), 962
ROUND, funzione (forma 1, data), 962
nell'aritmetica delle date, 172-173
ROUND, funzione (forma 2, numeri), 963
ROWID, pseudocolonna, 373, 964
ROWIDTOCHAR, funzione, 189, 195, 964
ROWNUM, pseudocolonna, 323-324,
 709-710, 964
 con order by, 325-326
RPAD, funzione, 115, 118, 965
RR, formato delle date, 177
RTRIM, funzione, 115, 120, 307, 965
RUN, comando, 965
Ruoli, 353, 961
 aggiunta di password, 365-366
 attivazione e disattivazione, 366-367
 concessione, 364-365
 creazione, 363-364
 nel sistema UNIX, 366
 revoca, 356
 revoca dei privilegi, 367-368
 rimozione di password, 366
scaricamento, 367-368
standard, 355
- S**
- Salvataggio a due stadi, 965
Salvataggio dello schermo, 281
SAVE, comando, 104-105, 279, 966
SAVEPOINT, comando, 966
SCAN (SQL*PLUS), 967
Scaricamento di tabelle, 339
Schemi, 437
 degli utenti, 532
SCORE, funzione di ConText, 560, 967
Seconda forma normale, 27
Segmento, 968
 di rollback, 414, 968
 temporaneo, 968
SELECT
 accesso, 361
 comando (forma 1, SQL), 968
 comando (forma 2, interno SQL), 970
 privilegio, 353
SELECT...INTO, istruzione, 971
Selezione di tipi di dati astratti, 82
SEQ, sinonimo, 621
Sequenze, 386-388, 972
 informazioni tramite le viste, 621
Server, 416
 ConText, 571-572
 DDL, 574
 DML, 574
 Linguistic, 574
 personalità, 571
 spegnimento, 573
Server manager, 972
Servizi linguistici, 586-588
SESSION_MAX_OPEN_FILES, parametro
 di init.ora, 516
Sessione, 972
SET, comando, 972
SET BUFFER, comando, 278, 280
SET CONCAT, comando, 314
SET CONSTRAINTS, comando, 979
SET DEFINE, comando, 314
SET ESCAPE, comando, 314
SET FEEDBACK, comando, 390
 negli aggiornamenti, 292-293
SET HEADING, comando, 390
SET HEADING OFF, comando, 270-271
SET HEADSEP, comando, 87
SET LINESIZE, comando, 88, 97

- SET LONG, comando, 526-527
 SET NEWPAGE, comando, 88, 98
 SET PAGESIZE, comando, 88, 98
 SET PAUSE, comando, 88, 104
 SET PAUSE OFF, comando, 104
 SET ROLE, comando, 367, 980
 SET SCAN, comando, 314
 SET SERVEROUTPUT, comando, 459
 SET SQLCASE UPPER, comando, 274
 SET TERMOUT OFF, comando, 272
 SET TRANSACTION, comando, 980
 SGA, 981
 Shared SQL Area, 452
 SHOW, comando, 45, 274, 981
 SHOW ALL, comando, 281
 SHOW BUFFER, comando, 279
 SHOW ERROR, comando, 641
 SHOW ERRORS, comando, 459
 SHOWMODE (SQL*PLUS), 982
 SHUTDOWN, 982
 procedura, 573
 SIGN, funzione, 140, 149, 982
 SIN, funzione, 140, 150, 982
 Sincronia, problemi, 444
 Singolarità, 687
 SINH, funzione, 140, 150, 983
 Sinonimi, 690, 982
 convalida, 474
 creazione, 361-362
 e link di database, 406-407
 e trasparenza delle locazioni, 410-411
 informazioni tramite le viste, 620-621
 per le procedure, 453
 pubblici, 370, 982
 Sintassi, 983
 Sistema a partizione condivisa, 983
 Sistema operativo, 983
 Sistema server, 983
 Sistemi orientati agli oggetti, differenze con i sistemi relazionali, 598
 Sistemi relazionali
 critiche, 18
 differenze con i sistemi orientati agli oggetti, 598
 SKIP, comando, 268
 SMON, processo, 983
 Snapshot, 531-532, 983
 aggiornamenti automatici, 541-542
 aggiornamenti manuali, 543-544
 basati su ROWID o chiavi primarie, 538-539
 creazione di indici, 536
 di sola lettura o aggiornabili, 534-535
 e trigger, 547-548
 gruppi di aggiornamento, 544-547
 indicizzazione delle tabelle, 540-541
 informazioni tramite le viste, 636-638
 log, 536, 539
 modifica, 551-552
 nomi, 539-540
 privilegi richiesti, 532-533
 scaricamento, 552-553
 semplici e complessi, 533-534
 visualizzazione di dati, 549-551
 SNP_n, processi, 542
 Somme, 97
 Sostituzione, 983
 Sottoquery, 983
 avanzate, 215-216
 caratteristiche, 61
 correlate, 218
 operatore where, 58-59
 restituzione di elenchi di valori, 60
 restituzione di valori singoli, 59
 Sottostringhe, ottenute tramite READ, 519-521
 Sottrazione, 143, 759
 SOUNDEX, funzione, 115, 136, 984
 ricerca di ConText, 566-567
 SP, suffisso per le date, 178
 SPACE (SQL*PLUS), 985
 Spazio libero, 985
 SPOOL, comando, 88, 98, 985
 SPOOL OFF, comando, 100, 281
 SPTH, suffisso per le date, 178
 SQL, 8-9, 39, 985
 buffer, 101, 278
 creazione di tabelle, 331-332
 distinzione con SQLPLUS, 90
 espressione CHR(12), 325
 file di avvio, 311-313
 funzione DECODE, 315-330
 SQL (*continua*)
 generatore di codice, 87
 operatore LIKE, 51
 script UTLEXCPT, 625
 selezione di dati, 42
 terminatore, 45
 SQL*LOADER, 584, 985
 SQL*NET, 404, 985
 descrittori di connessione, 635
 stringhe di connessione, 409-410
 SQL*PLUS, *vedere* SQLPLUS
 SQLCASE (SQL*PLUS), 985

- SQLCODE, 986
SQLCONTINUE (SQL*PLUS), 986
SQLERRM, istruzione, 986
SQLLOAD, comando, 986
SQLNUMBER (SQL*PLUS), 998
SQLPLUS, 39, 87, 389-401, 998
aggiunta di comandi, 107
avvio, 40
caricamento di variabili, 395-396
comandi di base, 87-88
comando copy, 414-415
comando describe, 617
comando password, 355
comando set, 45
conteggio delle righe, 44
controllo dell'ambiente, 108-111
conversioni in maiuscolo, 274
creazione di report, 92-101, 257-277
definizione dell'editor, 105
diagrammi a barre e grafici, 298-300
distinzione con SQL, 90
ed editor dell'utente, 89
editor della riga di comando, 90,
101-104
editor predefinito, 107
file di avvio, 89
impostazioni di default, 107
salvataggio dei comandi, 280-281
in un buffer, 278
stile dei comandi, 41,
uscita, 41
SQLPREFIX (SQL*PLUS), 999
SQLPROMPT (SQL*PLUS), 999
SQLTERMINATOR (SQL*PLUS), 999
SQRT, funzione, 140, 147
SS, formato delle date, 178
SSSSS, formato delle date, 178
Standard di denominazione, 20-22
START WITH, clausola
del comando create snapshot, 542
negli alberi genealogici, 248-251
START, comando, 88, 108, 279, 393, 999
STATUS, comando della procedura
 CTXCTL, 573
STDDEV, funzione, 140, 155, 1000
STOP ALL, comando della procedura
 CTXCTL, 573
STORAGE, clausola, 510, 1000
STORE, comando, 281, 1001
STRANO, tabella, 331
Stringhe, 114
 cancellazione di caratteri, 309
caratteri e date, 398
concatenazioni, 117
conteggio ricorrenze, 308-309
conversioni, 300-301
copia e incolla, 304-308
di caratteri, 114
di connessione, 409-410
funzioni, 115
informazioni sulle, 135
lunghezza, 125-126
ricerca delle simili, 136
ricerche, 129-133
ricorrenze, 130
ritaglio, 126-128
sostituzione di caratteri, 196
taglia e incolla, 118
tipo CHAR, 115
tipo VARCHAR2, 115
SUBMIT, funzione del package CTX_LING,
587
SUBSTR, funzione, 115, 126-128, 191, 1001
e indici, 376
SUBSTRB, funzione, 1002
SUBTRACT, procedura del package
 DBMS_REFRESH, 546
SUFFIX (SQL*PLUS), 1002
SUM, funzione, 140, 151, 1002
SVRMGR, 1002
SYN, sinonimo, 620
SYS, utente, 354, 1002
 e snapshot, 533
SYS_C#####, nomi standard dei vincoli, 338
SYSDATE, pseudocolonna, 164, 172, 1002
System Global Area (SGA), 452, 1003
SYSTEM
 tablespace, 1002
 utente ORACLE, 354, 1002
YYYYY, formato delle date, 177

T

- Tabelle, 5, 1003
annidate, 496, 1003
 esecuzione di query, 499-503
 inserimento di record, 497-503
 problemi, 503-506
accesso, 694-696
aggiornamento, 292-293, 340-341
aggiornamento di dati, 285-293
aggiunta di colonne, 339
aggiunta di nuove righe, 286
annidate, 73

- AZIONE, 375
chiavi candidate, 336
chiavi esterne, 337
chiavi primarie, 26, 337
 complete, 28
CLIENTE, 85
CLIMA, 11
colonne, 7-8
combinazioni, 62-63
COMFORT, 152
COMPLEANNO, 171
conteggio delle righe, 390
correlate, 11, 16, 29
creazione, 331-338
creazione a partire da altre tabelle, 345-346
creazione di indici, 349, 373-377
creazione di viste complesse, 241-243
CTX_TEMP, 585
di solo indice, 345, 1003
distinzione tra locali e remote, 407
DUAL, 165
e viste oggetti, 75
entità, 8
ereditarietà, 345
EXCEPTIONS, 625
FATTURA, 316
GIORNALE, 42
highwatermark, 288
HOCKEY, 372
impostazioni per ConText, 575-582
indipendenti, 11
INDIRIZZO, 127
informazioni, 8
informazioni sui commenti, 626
informazioni sui vincoli, 622-625
informazioni tramite le viste, 614-615
inserimento di dati, 285-288
inserimento di record, 81
inserimento di selezioni, 287
inserimento di valori NULL, 286
interrogazione, 64
LAVORATORE, 26
LOCAZIONE, 60
log di snapshot, 536
master, 534
MATEMATICA, 142
memorizzazione, 385
modifica, 339-341
modifica dei dati, 285
NOME, 304
nomi delle, 332
numero di indici, 378
oggetto, 1003
 aggiornamenti e cancellazioni, 592
 e OID, 590-591
 generazione dei riferimenti, 601-602
 inserimento di righe, 591-592
selezione di dati, 592
ordinamento, 10
PRESTITO, 490
problemi di denominazione, 33
PROPOSTA, 507, 508
REGISTRO, 94, 384
relazioni logiche, 63
relazioni multiple, 73
restrizioni agli accessi remoti, 412
ricerca con MAX e MIN, , 158
righe, 7-8
rimozione di righe, 291-292
RIVISTA, 131
salvataggio in cluster, 384-385
scaricamento, 339
segmenti, 381
selezione di tutte le colonne, 320
sicurezza, 368
STRANO, 331
suddivise in partizioni, 347-351
trasparenza delle locazioni, 410-411
trasposizioni, 321-323
troncamento, 339
unione, 216-217
unioni esterne, 224-225
utilizzo dell'operatore INTERSECT, 232-233
utilizzo dell'operatore UNION, 228-229
vincoli, 336
VIRGOLA1, 303
TABLE, tipo di dati PL/SQL, 1004
Tablespace, 354, 379-384, 1004
 blocchi, 1004
 estensioni iniziali, 381
 READ ONLY, 383
TABLESPACE
 clausola, 510
 opzione del comando create index, 379
TABS, sinonimo, 614
Taglia e incolla, 118
TAN, funzione, 140, 150, 1004
TANH, funzione, 140, 150, 1005
TCP/IP, 404
Tema, 1005

- TEMPORARY, parola chiave del comando
create tablespace, 383
- Terminale
asincrono, 1005
sincrono, 1005
- Terminatore SQL, 45, 1005
- TERMOOUT (SQL*PLUS), 1005
- Terza forma normale, 28
- Test logici
coordinazione, 221-222
if, then, else, 315-316
- Testi
aggiunti ai database, 555-557
ricerche tramite ConText, 558-568
- TEXT_ENABLE, parametro, 574-575
- TH, suffisso per le date, 178
- THE, parola chiave; per le tabelle annidate, 499-501
- THSP, suffisso per le date, 178
- TIME (SQL*PLUS), 1005
- TIMING (forma 1, SQL*PLUS), 1005
- TIMING (forma 2, SQL*PLUS), 1006
- Tipi di dati, 853
alfanumerici, 114
astratti
indicizzazione attributi, 474
informazioni tramite le viste, 632-635
selezione, 82
sicurezza
attributi, 83
conversione tra, 187-195
costruttori, 81
direttive per la conversione, 194
elenco, 114
- TITLE, comando, 87, 93
- TITLE OFF, comando, 270-271
- Titoli
formattazione, 262
impostazione, 93
utilizzo di variabili, 397
- TNSNAMES.ORA, file, 409-410
- TO_BINARY_INTEGER, funzione, 1006
- TO_CHAR, funzione, 166, 173-177, 189-190, 304
- TO_DATE, funzione, 101, 166, 173-177, 180, 189-190, 286, 1007
- TO_LABEL, funzione, 1008
- TO_MULTI_BYTE, funzione, 189, 195, 1008
- TO_NUMBER, funzione, 189-190, 1008
- TO_SINGLE_BYTE, funzione, 189, 195, 1008
- Transazione, 1008
- TRANSLATE, funzione, 189, 196, 300-303, 1009
- Trasparenza delle locazioni, 410-411, 417
- Trasposizioni di tabelle, 321-323
- Trigger, 437, 1009
a livello di istruzione, 439
a livello di riga, 438
- AFTER ROW, 549
attivazione e disattivazione, 447-449
- BEFORE e AFTER, 439
- combinazioni, 442-444
- compilazione manuale, 449
- dati duplicati, 444-445
- e snapshot, 547-548
- eliminazione, 449
- impostazione di valori, 444
- informazioni tramite le viste, 639-640
- INSTEAD OF, 439-440, 480-486
- nomi, 447
- privilegi necessari, 437-438
- sintassi, 440-442
- sostituzione, 449
- TRIM, procedura di DBMS_LOB, 515, 528
- TRIMOUT (SQL*PLUS), 1009
- Troncamento di tabelle, 339
- TRUNC, funzione (forma 1, date), 1009
- TRUNC, funzione (forma 2, numeri), 1010
- TRUNCATE, comando, 288, 292, 339, 1011
- TTITLE, comando, 262-264, 1011
- TTITLE OFF, comando, 110
- Tupla, 8, 1012
- TYPE, funzione (interno SQL), 1012
- U**
- UID, pseudocolonna, 1012
- UNDEFINE, comando, 1012
- UNDERLINE (SQL*PLUS), 1013
- Unicità degli indici, 374-375
- UNION, 710-714
funzione, 1013
operatore, 228-235
controllo delle ripetizioni, 232
restrizioni, 237-239
- UNION ALL, operatore, 230
- Unione, 720-732
di hash, 1013
di tabelle, 216-217
- Unioni
di colonne, 213
di elenchi, 228-233
esterne, 224-225

- UNIQUE, vincolo, 337
 Unità logiche di lavoro, 1013
UNLIMITED TABLESPACE, privilegio, 370, 532-533
UNRECOVERABLE, parola chiave, 1013
UPDATE, comando, 292-293, 340, 407
 con NULL, 295
 con select incorporata, 294-295
 problemi, 295
UPDATE, istruzione (forma 1, interno SQL), 1014
UPDATE, istruzione (forma 2, PL/SQL), 1014
UPDATE, comando (forma 3, comando SQL), 1015
UPDATE, privilegio, 368
UPDATE_POLICY, procedura del package CTX_DDL, 578
UPI (USER PROGRAM INTERFACE), 1016
UPPER, funzione, 115, 124-125, 1016
USER, pseudocolonna, 411-413, 1016
USER PROGRAM INTERFACE (UPI), 1016
USER_CATALOG, vista, 612
USER_CLU_COLUMNS, vista, 631
USER_CLUSTER, vista, 630-631
USER_COLL_COMMENTS, vista, 626-627
USER_COLL_TYPES, vista, 491, 634
USER_CONS_COLUMNS, vista, 624
USER_CONSTRAINTS, vista, 622-623
USER_DB_LINK, vista, 635-636
USER_DIRECTORY, vista, 635
USER_ERRORS, vista, 459, vista, 641
USER_IND_COLUMNS, 627-629
USER_INDEX, vista, 627-628
USER_INDEXES, dizionario, 373
USER_LOB, vista, 634-635
USER_METHODS_RESULT, vista, 633
USER_OBJECT_SIZE, vista, 642
USER_OBJECTS, vista, 613
USER_REFRESH, vista, 544, vista, 638
USER_REFRESH_CHILDREN, vista, 544, 638
USER_REFS, vista, 634
USER_SEQUENCE, vista, 621
USER_SNAPSHOTS, vista, 549-550, 636-638
USER_SNAPSHOTS_LOGS, vista, 551, 638-639
USER_SOURCE, vista, 467, 640-641
USER_SYNONYMS, vista, 620-621
USER_TAB_COL_STATISTICS, vista, 616
USER_TAB_COLUMNS, vista, 79, 489, 616-617
USER_TAB_COMMENTS, vista, 626
USER_TAB_HISTOGRAMS, vista, 617-618
USER_TABLES, vista, 389, 614-615
USER_TRIGGERS, vista, 639-640
USER_TYPE_ATTRS, vista, 80, 492
USER_TYPE_METHODS, vista, 633
USER_TYPE_METHODS_PARAMS, vista, 633
USER_TYPES, vista, 491, 632
USER_TYPES_ATTRS, vista, 632
USER_UPDATABLE_COLUMNS, vista, 618
USER_VIEWS, vista, 618-619
USERENV, funzione, 1016
USING INDEX, clausola, 379
Utenti, 353
 cancellazione, 357
 concessione di privilegi, 356-357
 creazione, 354
 ID, 40
 passaggi ad altri, 359-361
Utilizzo delle funzioni, 805
UTLEXCPT.SQL, script, 625
- ## V
- VA**, suffisso, 490
Valori assoluti, 145
Valori singoli, 139
 restituzione, 59
 sostituzione, 197
 test logici, 55
Valori sostituti, 145
VALUE, operatore, 596-597
VALUES, parola chiave, 286
VAR, funzione (interno SQL), 1017
VARCHAR, tipo di dati, 1017
VARCHAR2, tipo di dati, 114, 187, 557, 1017
Variabili
 caricamento, 395-396
 conversioni in maiuscolo, 274
 dichiarazione (PL/SQL), 421, 1017
 elenco di quelle definite, 275
 modifica, 396
 nei file di avvio, 313-314
 sostituzioni registrate, 309-311
 utente, 1018
 utilizzo con ttitle, 397
VARIABLE, funzione, 1017

VARIANCE, funzione, 140, 155, 1018
Varianza, 155
VARRAY, clausola, 74, 1018
VERIFY (SQL*PLUS), 274, 1018
Versione di revisione, 1018
Vincoli
assegnazione dei nomi, 338
di colonna, 336
di controllo, 338
di integrità referenziale, 337
di tabella, 336
di unicità, 346
informazioni tramite le viste, 622-625
VIRGOLA, tabella, 303
Virgole, rimozione, 302-303
Viste, 1020
aggiunta, 262-263
alias, 208
clausola from, 255-256
clausola order by, 343
combinata, 244-246
complesse, creazione, 241-243
costruzione dinamica, 323
creazione, 64-65, 341-343
del dizionario di dati, 1020
di gruppi, 205-206, 243
potenzialità, 209-210
di sola lettura, creazione, 344-345
dei totali, 243-244
e link di database, 406-407
espansione della visualizzazione, 66
informazioni tramite le viste, 618-620
limitazioni, 342
manipolazione, 65
ordinamento con group by, 326
pseudocolonna User, 411-413
query sulle, 322
selezione di tutte le colonne, 320
stabilità, 342
Viste oggetto
con REF, 598-600
elaborazione di dati, 479
implementazione, 476-479
query, 602

Visualizzazione di codici sorgenti, 467
VSAM, 22
VSIZE, funzione, 140, 1026
W
W, formato delle date, 178
WHEN, clausola del comando create trigger, 441
WHEN OTHERS, clausola, 436, 463
WHENEVER, comando (forma 1, interno SQL), 1026
WHENEVER SQLERROR, comando (forma 2, SQL*PLUS), 1027
WHERE, clausola, 45-47, 84, 1028
 con le date, 182-183
 con SOUNDEX, 136
WHILE, cicli, 432-433
WITH ADMIN OPTION, clausola del comando grant, 356
WITH GRANT OPTION, clausola del comando grant, 357
WITH OBJECT OID, clausola del comando create view, 600-601
WITH PRIMARY KEY, clausola, 541, 549
WITH READ ONLY, clausola del comando create view, 344
WITH ROWID, clausola, 541, 548
WNPS, restrizione, 485
WORD_WWRAPPED, comando, 95
WRAP (SQL*PLUS), 1029
WRITE, procedura di DBMS_LOB, 515, 524-525
WW, formato delle date, 178

Y

Y, formato delle date, 177
YEAR, formato delle date, 177
YY, formato delle date, 177
YYY, formato delle date, 177
YYYY, formato delle date, 177